

Tmax プログラミングガイド (UCS)

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 プログラミングガイド (UCS)

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. プログラムのフロー	1
1.1.1. TCS	1
1.1.2. POD	2
1.1.3. UCS	2
1.2. UCSとRDP	2
1.2.1. UCS	2
1.2.2. RDP	4
第2章 UCSプログラム	5
2.1. 概要	5
2.2. UCSプログラムの構成	5
2.2.1. サーバー・プログラム	7
2.2.2. クライアント・プログラム	10
2.3. UCS環境設定およびコンパイル	10
2.3.1. 環境設定	10
2.3.2. コンパイル	11
第3章 UCS関数	13
3.1. 概要	13
3.2. サーバー/クライアント関数	14
3.2.1. tppost	14
3.2.2. tpsetfd	15
3.2.3. tpunsubscribe	18
3.2.4. tpunsubscribe	20
3.3. サーバー関数	21
3.3.1. tpclrfd	21
3.3.2. tpissetfd	23
3.3.3. tpregcb	25
3.3.4. tprelay	27
3.3.5. tpsavectx	29
3.3.6. tpschedule	32
3.3.7. tpsendtocli	33
3.3.8. tpsvctimeout	35
3.3.9. tpsvrdone	36
3.3.10. tpsvrdown	38
3.3.11. tpunregcb	39
3.4. クライアント関数	40
3.4.1. tpgetunsol	40
3.4.2. tpsetunsol	43

3.4.3. tpsetunsol_flag	45
第4章 UCSの使用例	47
4.1. プログラムの異常終了管理	47
4.2. ソケットを使用した非同期通信プログラム	50
4.3. RQを使用したプログラム	55
4.4. クライアント対サーバー・プログラム	57
4.5. サーバー対サーバー・プログラム	63
第5章 RDPプログラム	71
5.1. 概要	71
5.2. RDPサーバー・プログラムの構成	71
5.3. RDP環境設定およびコンパイル	72
5.3.1. 環境設定	72
5.3.2. コンパイル	72
第6章 RDPの使用例	75
6.1. クライアント対サーバー・プログラム	75
索引	83

図目次

[図 1.1]	TCSタイプのサーバー・プログラム	1
[図 1.2]	UCSサーバー・プログラム	2
[図 1.3]	RDPの動作方式	4
[図 2.1]	UCSサーバー・プログラムの構成	8

このガイドについて

対象読者

本書は、Tmax[®](以下、Tmax)サーバー・プロセスのうち、Tmax UCSについての概念と機能および特性について説明しています。同書は、Tmax UCSを用いて開発する開発者とシステム管理者を対象としています。

前提知識

本書は、Tmaxシステムの概要とTmaxシステムが提供する各種機能や特性などを習得するための手引書です。

本書を理解するためには、以下の事項についての知識が必要です。

- ミドルウェアおよびUNIXシステムについての基本知識
- Tmaxの基本概念
- JavaとCプログラミングについての知識

制限事項

本書では、UCSに関連する関数についてのみ説明しています。実務上の具体的な使用方法や管理・運用についての内容は、各製品ガイドを参照してください。

参考

Tmaxシステムの開発についての基本的な内容は、『Tmax 運用ガイド』および『Tmax アプリケーション開発ガイド』を、Tmaxが提供するコマンドとC API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

本書は、計6章で構成されています。

各章の主な内容は以下のとおりです。

- 第1章: 紹介

UCSの概要および関数について説明します。

- 第2章: UCSプログラム

UCSプログラムの構成、環境設定、コンパイル方法について説明します。

- 第3章: UCS関数

UCSで使用する関数について説明します。

- 第4章: UCSの使用例

UCSプログラムの使用例について説明します。

- 第5章: RDPプログラム

RDPプログラムの構成、環境設定、コンパイル方法について説明します。

- 第6章: RDPの使用例

RDPプログラムの使用例です。

表記上の規約

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
<div>参考</div>	参照/注意事項
[図1.1]	図の名前
[表1.1]	表の名前
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

関連文書

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、UCSの概要および関数について説明します。

1.1. プログラムのフロー

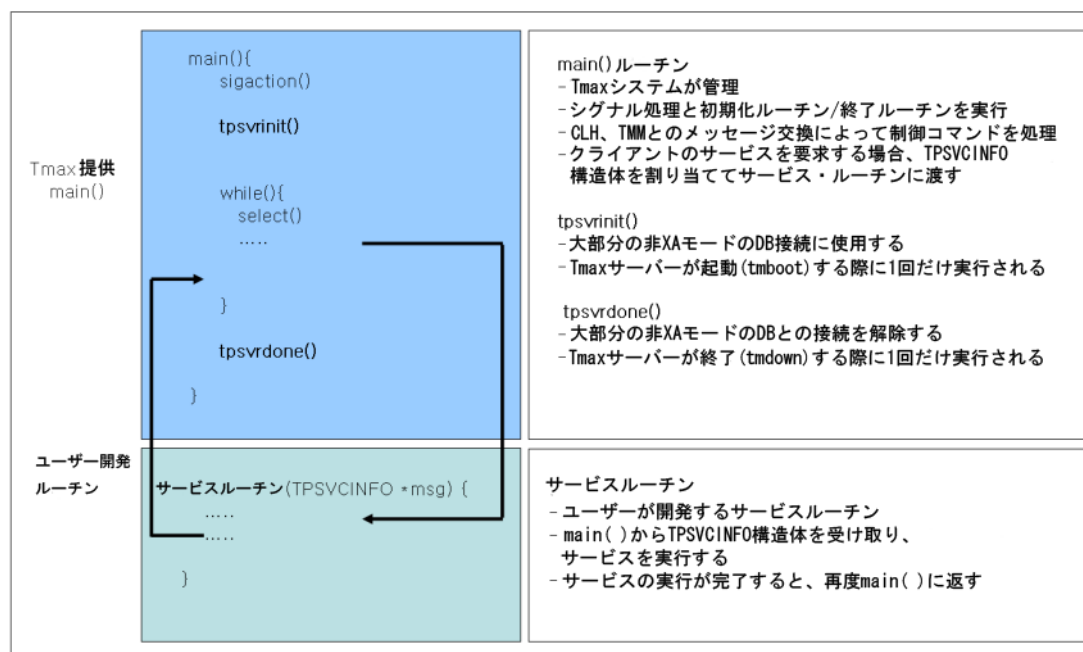
1.1.1. TCS

TCS(Tmax Control Server)は、他のミドルウェア(Tuxedo、Entera、TopEndなど)で使用されるように、クライアントの要求がある場合にのみサービスを提供する典型的なプロセス制御方法です。

TCSプロセスは、一般的な3階層(Tier)の概念を持っているサーバー・プロセスです。Tmaxはクライアントの要求を適切なサーバー・プロセスのサービスに渡し、サーバー・プロセスのサービスはビジネス・ロジックに適切な演算処理を行った後、結果値を作成してクライアントに返します。

TCSタイプのサーバー・プログラムのフローは以下のとおりです。

[図 1.1] TCSタイプのサーバー・プログラム



1.1.2. POD

POD(Process On Demand)は、クライアントの要求時にサーバー・プロセスが起動してサービスを実行するプロセス制御方法です。

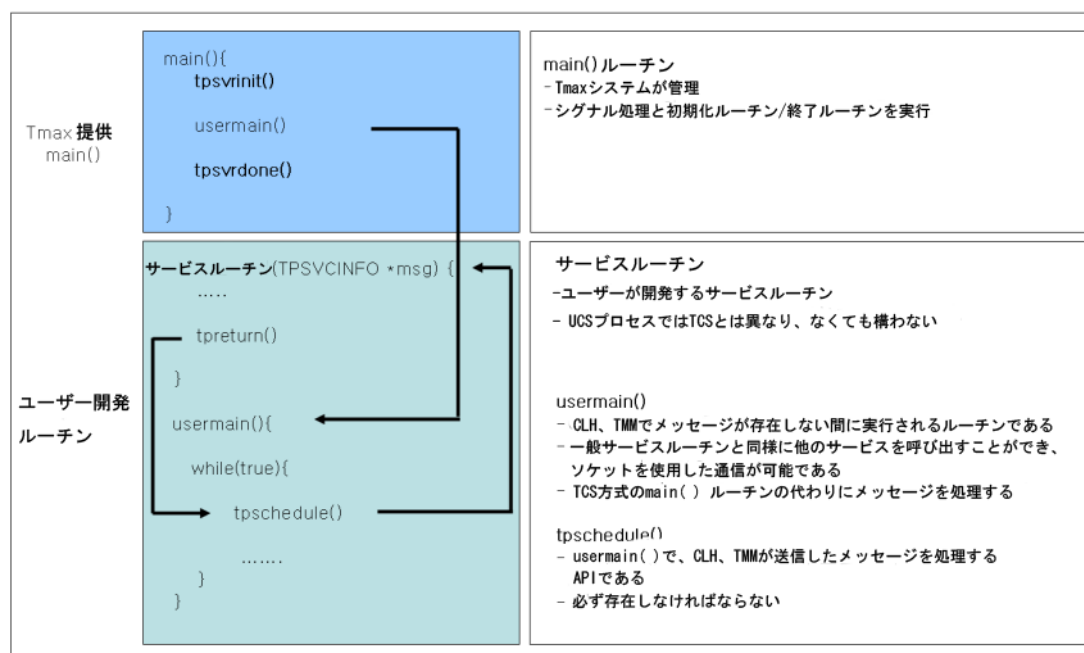
PODプロセスは、TCSと類似の形式を持つプロセスです。Tmaxを起動する際にプロセスが起動していなくても、クライアントから要求が来るとTmaxで該当プロセスを起動させ、要求を処理後に再度プロセスを終了させる点がTCSと異なります。

1.1.3. UCS

一般的にTCSプロセスはクライアントの要求に対してのみ処理を行います。一方、UCS(User Control Server)プロセスはクライアントの要求がなくてもクライアントにデータを送信などの処理を行うことが可能であり、また、クライアントの要求にも対応します。UCSは、TPモニター製品の中でTmaxだけの固有機能です。

UCSタイプのサーバー・プログラムのフローは以下のとおりです。

[図 1.2] UCSサーバー・プログラム



1.2. UCSとRDP

1.2.1. UCS

UCS(User Control Server)とは、プログラム処理のフローをミドルウェアで制御せず、ユーザーが直接制御できるTmax固有のプログラム形式です。これを使用したプログラムとプロセスを、それぞれUCSプログラム

およびUCSプロセスといいます。クライアントの要求(イベント)によってのみ処理を行う既存の方式に加え、サーバー・プロセス(UCSプロセス)が能動的にイベントを発生させ、処理を行うことができます。

クライアントの要求がなくても情報伝達が必要な業務(為替レート情報、notifyなど)、作業スケジューリング業務、および対外機関との連動業務などにUCSが使用されています。

UCSの必要性

UCSはユーザーが制御できるサーバーです。従来デーモン形式で起動して実行していた特定の時間に、作業の実行、他の対外機関との連動作業、クライアントへの情報伝達などをUCS形式で作成できます。このように作成されたプログラムのプロセスはすべてTmax管轄になり、プロセス管理(異常終了の場合の再起動など)は別途必要ありません。

以下は、UCSが使用されるケースです。

- 情報を変更すると、接続されているクライアント(特定のクライアントやすべてのクライアント)に通知する形式の業務

- 業務事例1

クライアントの要求がなくても、証券業務で時々刻々と変動する為替レート情報を伝達し、よりスピーディーで正確なデータをクライアントは受け取ることができます。

- 業務事例2

ソフトウェア障害や作業日程についてのメッセージ、その他告知事項などを、管理者や開発者、またはユーザーに通知できます。

- 業務事例3

銀行業務で為替取引が発生した場合、該当クライアントに直接通知したり、ATMの状態を特定のクライアント(ATM管理者)に通知したりすることができます。

- 作業スケジューリング形式の業務

- 業務事例

特定の時間にデブロイ作業を実行したり、その他スケジューリングが必要な業務に使用したりできます。

- 対外連動業務

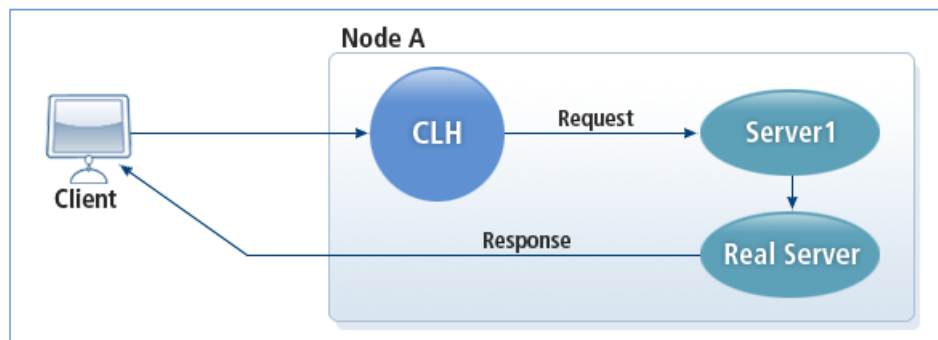
- 業務事例

対外機関との連動業務を実装する際に、より効率的で簡単なプログラム構造を提供します。非同期形式の取引フロー、データの到着有無をチェックするためのlooping(以下、ルーピング)がなくなるため、プログラムを簡単に作成できるだけでなく、CPUのリソースを節約できます。また、多様な形式の業務をUCSタイプで実装できます。したがって、開発者がUCSプログラムの処理フローを正確に理解して使用すると、業務を実装する際に有効に使用できます。

1.2.2. RDP

RDP(Realtime Data Processor)はUCSと同じ形式のサーバーで動作します。RDPは、持続的に変化するデータを効率的かつスピーディーにクライアントに伝達するために、UCSタイプのプロセスをカーネルレベルで改良したプロセスです。RDPはCLHを経由せずにクライアントにデータを伝達するため、少量のデータを多数のクライアントに短時間の間隔で送信した場合に、プロセス占有率や処理速度においてUCSより優れた性能を発揮します。

[図 1.3] RDPの動作方式



RDPサーバーは1つのノードに一意である必要があります。該当ノードに存在するすべてのサービスは、RDPサーバーにサービスの結果データを送信した後、RDPサーバーからデータをクライアントに送信します。

他のサーバー・プロセスで処理した結果を取得するためのチャンネル数(RSCPC)を設定する必要があり、CLHとRDPサーバー・プロセスの数は一定に維持される必要があります(MIN値とMAX値が同じである必要があります)。ただし、RDPサーバー・プロセスのMIN、MAX値は常にCLHより大きい必要があり、通常は2倍数に設定します。また、CLHはMINCLH、MAXCLHの数を同一に設定する必要があります。

プログラム構成およびAPIはUCSと同一で、環境設定とコンパイル方法は若干異なります。RDPの環境設定およびコンパイル方法についての詳細は[「5.3. RDP環境設定およびコンパイル」](#)を参照してください。

第2章 UCSプログラム

本章では、UCSプログラムの構成、環境設定、コンパイル方法について説明します。

2.1. 概要

UCS(User Control Server)とは、プログラム処理のフローをミドルウェアで制御せず、ユーザーが直接制御できるTmax固有のプログラム形式です。これを使用したプログラムとプロセスを、それぞれUCSプログラムおよびUCSプロセスといいます。

クライアントの要求がなくても情報伝達が必要な業務(為替レート情報、notifyなど)、作業スケジューリング業務、および対外機関との連動業務などにUCSが使用されています。

2.2. UCSプログラムの構成

UCSプログラムは、`usermain()`の本体が存在する必要があり、コンパイルする際にUCSライブラリー(`libsvrucs.a`/`libsvrucs.so`)とリンクする必要があります。基本的に`usermain()`は無限ループ形式で実装されます。`usermain()`が終了(関数リターン)すると、該当プロセスは`tpsvrdone()`を実行し、終了するためです。`usermain()`は、`main()`が終了すると該当プロセスが終了するのと同様です。

注

UCSプログラムを作成する際の注意点は、クライアントから送信されたデータやTmaxカーネルから送信された終了コマンドは`tpschedule()`を通じて受け取るため、`usermain()`の無限ループ内に`tpschedule()`が常に存在しなければならないことです。

以下は、接続されたクライアントに特定情報を非要求メッセージ形式で送信する`usermain()`の簡単な例です。

```
+1  #include <stdio.h>
+2  #include <usrinc/atmi.h>
+3  #include <usrinc/ucs.h>
+4
+5  #define MAX_CLI 10
+6  int num_cli;
+7  int client_id[MAX_CLI];
+8  int count;
+9
+10 int
+11 tpsvrinit(int argc, char *argv[])
```

```

+12 {
+13     num_cli = 0;
+14     count = 0;
+15 }
+16
+16 int
+17 usermain(int argc, char *argv[]) /* Tmaxのucsモードでmainと同じ部分 */
+18 {
+19     int        jobs;
+20     int        i;
+21     int        ret;
+22     char        *sndbuf;
+23     static int count = 0;
+24
+25     printf("usermain start\n");
+26
+27     sndbuf = (char *)tpalloc("CARRAY", NULL, 1024);
+28
+29     while(1) {
+30         for (i = 0; i < num_cli; i++) {
+31             sprintf(sndbuf, "Success tpsendtocli [%d] 012345678901234\n",
+32                     count);
+33             tpsendtocli (client_id[i], sndbuf, 1024, 0);
+34             count++;
+35             sleep(1);
+36             jobs = tpschedule(-1);
+37         }
+38     }
+39
+40     LOGIN(TPSVCINFO *msg) /* Tmax SERVICE部分 */
+41     {
+42         char        *sndbuf;
+43         int        cliid;
+44         int        ret;
+45         int        i;
+46
+47         printf("msg->data = [%.s]\n", msg->len, msg->data);
+48         fflush(stdout);
+49
+50         sndbuf = (char *)tpalloc("CARRAY", NULL, 1024);
+51
+52         printf("Success transaction");
+53         sprintf(sndbuf, "Success transaction");
+54
+55         if (num_cli < MAX_CLI) {

```



```

+56         client_id[num_cli] = tpgetclid();
+57         printf("client id(clid) = %d\n", client_id[num_cli]);
+58         num_cli++;
+59     }
+60
+61     tpreturn(TPSUCCESS, 0, (char *)sndbuf, 1024, 0);
+62 }

```

以下は、サンプルプログラムについての簡単な説明です。

1. 初めて起動する際にtpsvrinit()を実行し、usermain()に入ります。
2. 30行目にて、num_cli値が0であるため、for文から抜けます。
3. 36行目にて、tpschedule()を使用して、クライアントからサービス要求があるのか、またはtmdownコマンドを受けたのかを確認します。
4. 40行目にて、クライアントからLOGIN SERVICE要求が来た場合に実行します。要求がない場合、tpschedule()から抜けます。
5. 56行目にて、tpgetclid()を使用して、接続されているクライアント情報をグローバル変数であるclient_idに保存した後、num_cli値を増加させます。
6. 61行目にて、LOGIN SERVICEを要求したクライアントにレスポンス・データを送信します。
7. 36行目にて、tpschedule()から抜けます。
8. 30行目にて、num_cli値が正数値であるためfor文を実行します。
9. 32行目にて、tpsendtocli()を使用して、client_idに保存されているクライアントに該当情報を送信します。
(この部分が非要求メッセージを送信する部分です)
- 10 3～9段階を繰り返し実行します。
11. tpschedule()でtmdownコマンドを受けた場合、tpsvrdone()を実行した後にプロセスが終了します。

2.2.1. サーバー・プログラム

UCSサーバー・プログラムは、以下のようなモジュールとライブラリーから構成されます。

- \$(TMAXDIR)/lib/libsvrucs.aまたは\$(TMAXDIR)/lib/shared/libsvrucs.so

プログラムのmain()と、各種UCS関連のAPIなどを含むライブラリーです。UCSプログラムを作成する際、常にリンクする必要があります。

- int tpsvrinit(int argc, char *argv[])

プログラムを起動する際に1回実行されます。グローバル変数の初期化や非XAの場合にデータベース接続などを実装します。

- int tpsvrdone()

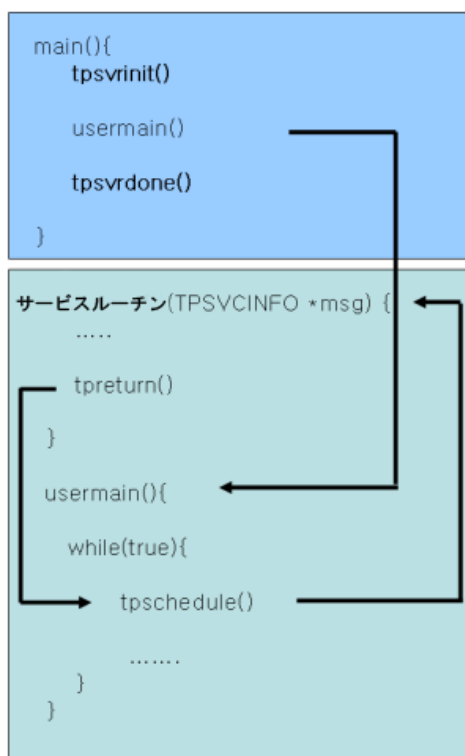
プログラムを終了する際に1回実行されます。使用リソースのリターンや非XAの場合にデータベース接続の解除などを実装します。

- int usermain(int argc, char *argv[])

実際にアプリケーションのロジックが実装される部分です。usermain()のモジュールで返されると、tpsvrdone()を実行し、プロセスが終了する形式です。そのため、大部分が無限ループ形式で実装されます。

クライアントの要求データやtmdownコマンドを受けるために、tpschedule()が常に必要です。tpschedule()が呼び出されなければTmaxサーバーは終了しません。tmdown -i形式で終了します。

[図 2.1] UCSサーバー・プログラムの構成



上記のように、UCSプログラムは1つのモジュールと1つのライブラリーから構成され、usermain()のモジュール内には必ずtpschedule()を呼び出す必要があります。

UCSプログラムは、上記のようなプログラム構成によって、ユーザーが能動的に制御できます。これを利用して、多様な業務を実行できる様々なAPIを提供します。

APIのプロトタイプは以下のパスに定義されています。

```
$(TMAXDIR)/usrinc/ucs.h
```

主なAPIのプロトタイプは以下のとおりです。各関数についての詳細は、「[第3章 UCS関数](#)」を参照してください。

- すべてのUCSプログラムで使用されるAPI
 - int tpschedule(int sec)
 - int tpuschedule(int usec)
- 非要求メッセージの送信に使用されるAPI
 - int tpsendtocli(int clid, char *data, long len, long flags)
 - int tpgetclid(void)
- 非Tmaxとの非同期通信のためのAPI
 - int tpsetfd(void)
 - int tpissetfd(void)
 - int tpclrfd(void)
 - int tpgetctx(CTX_T *ctxp)
 - int tpcancelctx(CTX_T *ctxp)
 - CTX_T *tpsavectx()
 - int tprelay(char *svc, char *data, long len, long flags, CTX_T *ctxp)
- UCS内でtpacall(非同期通信)関数を使用する際に使用されるcallback API
 - int tpregcb(void)
 - int tpunregcb(void)

2.2.2. クライアント・プログラム

UCSプログラムからサービスが提供されるクライアント・プログラムの場合、一般的なプログラムとほぼ同じ形式で、以下の2つの設定を含む必要があります。

- UCSプログラムから送信されるデータを受信する設定
- UCSプログラムから送信されるデータを処理する設定

クライアント・プログラムで使用する関数のうち、UCSプログラムから送信されるデータを受信できるように設定する関数は以下のとおりです。

- `tpsetunsol_flag()`

`tpstart()`を使用してTmaxと接続する際に使用したフラグ値を再設定する関数です。通常、非要求メッセージを受信するか否かのフラグを設定します。

- `tpsetunsol()`

クライアントで使用する関数で、受信した非要求メッセージを処理するルーチンを設定する関数です。

- `tpgetunsol()`

クライアントで使用する関数で、非要求メッセージの受信関数です。

各関数についての詳細は「[第3章 UCS関数](#)」を参照してください。

2.3. UCS環境設定およびコンパイル

2.3.1. 環境設定

環境設定ファイルのSERVERセクションにSVGNAMEを設定後、SVRTYPEにUCSを指定します。

以下はUCS環境ファイルの例です。

```
*DOMAIN
tmax1          SHMKEY =79970, MINCLH=1, MAXCLH=3,
TPORTNO=8844, BLOCKTIME=120, RACPORT=3443
*NODE
tmaxs1  MAXDIR = "/user1/jaya/tmax3511",
        APDIR  = "/user1/jaya/tmax3511/appbin",
        PATHDIR = "/user1/jaya/tmax3511/path",
        TLOGDIR = "/user1/jaya/tmax3511/log/tlog",
        ULOGDIR = "/user1/jaya/tmax3511/log/ulog",
        SLOGDIR = "/user1/jaya/tmax3511/log/slog"
```

```

tmaxs2  TMAXDIR = "/user/jaya/tmax3511",
        APPDIR  = "/user/jaya/tmax3511/appbin",
        PATHDIR = "/user/jaya/tmax3511/path",
        TLOGDIR = "/user/jaya/tmax3511/log/tlog",
        ULOGDIR = "/user/jaya/tmax3511/log/ulog",
        SLOGDIR = "/user/jaya/tmax3511/log/slog"

*SVRGROUP
svg1          NODENAME = "tmaxs1"
svg2          NODENAME = "tmaxs2"

*SERVER
ucssvr1       SVGNAME = svg1,
              SVRTYPE = UCS,
              CPC = 5          # UCSとCLH間のチャンネル数
ucssvr2       SVGNAME = svg2,
              SVRTYPE = UCS

*SERVICE
SVC1          SVRNAME = ucssvr1

```

2.3.2. コンパイル

UCSサーバー・プログラムをコンパイルする場合、UCSライブラリー(libsvrucs.aあるいはlibsvrucs.so)とリンクする必要があります。

プログラム内でも\$TMAXDIR/usrinc/ucs.hが含まれている必要があり、Makefile内のTMAXLIBSに**-lsvrucs**を含む必要があります。

以下は、32bit SolarisでUCSサーバー・プログラムをコンパイルするためのMakefileの例です。

```

# Server makefile
TARGET   = $(COMP_TARGET)
APOBJS   = $(TARGET).o
SDLFILE  = demo.s

# Solarisの場合
LIBS      = -lsvrucs -lsocket -lnsl -nodb

# 他のOSの場合
LIBS      = -lsvrucs -nodb
OBSJ     = $(APOBJS) $(SDLOBJ) $(SVCTOBJ)
SDLOBJ    = ${SDLFILE:.s=_sdl.o}
SDLC      = ${SDLFILE:.s=_sdl.c}
SVCTOBJ   = $(TARGET)_svctab.o
CFLAGS    = -O -I$(TMAXDIR)

```

```
APPPDIR = $(TMAXDIR)/appbin
SVCTDIR = $(TMAXDIR)/svct
LIBDIR = $(TMAXDIR)/lib

#
.SUFFIXES : .c
.c.o:
$(CC) $(CFLAGS) -c $<
#
# server compile
#
$(TARGET): $(OBJS)
    $(CC) $(CFLAGS) -L$(LIBDIR) -o $(TARGET) $(OBJS) $(LIBS)
    mv $(TARGET) $(APPPDIR)/.
    rm -f $(OBJS)
$(APOBJS): $(TARGET).c
    $(CC) $(CFLAGS) -c $(TARGET).c
$(SVCTOBJ):
    touch $(SVCTDIR)/$(TARGET)_svctab.c
    $(CC) $(CFLAGS) -c $(SVCTDIR)/$(TARGET)_svctab.c
$(SDLOBJ):
    $(TMAXDIR)/bin/sdlc -i ../sdl/$(SDLFILE)
    $(CC) $(CFLAGS) -c ../sdl/$(SDLC)
#
clean:
    -rm -f *.o core $(TARGET)
```

参考

OSに応じてMakefileの内容が異なることがあります。

第3章 UCS関数

本章では、UCSで使用する関数について説明します。

3.1. 概要

以下は、UCSプログラムで使用する関数の一覧です。

- サーバー/クライアント関数

関数	説明
tppost	特定イベントを発生させ、メッセージを送信します
tpsetfd	ソケットfdをUCSプロセスのスケジューラーに登録します
tpsubscribe	特定イベントのメッセージの要求に登録します
tpunsubscribe	特定イベントのメッセージの登録を解除します

- サーバー関数

関数	説明
tpclrfd	プロセス内部のfdsetのソケットfdをオフします
tpissetfd	ソケットfdにデータが到着したのかをチェックします
tpregcb	非同期要求に対する応答を受けるルーチンを設定します
tprelay	サーバー・プロセスでサービスを要求したクライアントの情報を保存して、また他のサービスを要求します
tpsavectx	サーバー・プロセスでクライアントの情報を内部的に管理します
tpschedule	サーバー・プロセスでデータの到着を待機します
tpsendtocli	指定されたクライアントに非要求メッセージを送信します
tpsvctimeout	サービス・タイムアウトが発生した場合に呼び出します
tpsvrdown	サーバー・プロセスを終了します
tpunregb	非同期要求に対する応答を受けるルーチンを再設定します

- クライアント関数

関数	説明
tpgetunsol	非要求メッセージを受信します

関数	説明
tpsetunsol	受信した非要求メッセージを処理するルーチンを設定します
tpsetunsol_flag	非要求メッセージの受信フラグを変更します

3.2. サーバー/クライアント関数

3.2.1. tppost

サーバーとクライアントで特定イベントを発生させ、メッセージを渡す関数です。tppost()は、名前がeventnamのイベントにtpsubscribe()で登録されたすべてのクライアントとサーバー・プロセスにイベントの発生を通知し、必要な場合はメッセージを渡します。

- プロトタイプ

```
# include <tmaxapi.h>
int tppost(char *eventname, char *data, long len, long flags)
```

- パラメータ

パラメータ	説明
eventname	NULLで終わる63文字以内の文字列で発生するイベントの名前を意味します。ワイルドカードやpartial-matchingなどはサポートしていません
data	送信メッセージに対するポインターです。tpalloc()によって割り当てられたバッファである必要があります
len	送信するバッファ長です <ul style="list-style-type: none"> – dataの長さの指定が不要なバッファを指す場合、lenは無視され、0が使用されます – dataの長さの指定が必要なバッファを指す場合、lenは0になれません – dataがNULLの場合、lenは無視されます
flags	現在はTPNOTIMEのみ意味があります

- 戻り値

戻り値	説明
正数	関数の呼び出しに成功した場合です。
負数	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

tppost()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。クライアントでは、ネットワーク・エラーが最も多いです
[TPEOS]	OSレベルのエラーが発生した場合です。メモリーの割り当てなどの問題を含みます
[TPEINVAL]	設定された引数が正しくない場合です
[TPENOENT]	tpsubscribe()の場合、TPEVCTL構造体のqname、あるいはsvclに該当するRQかサーバーが存在しない場合です
[TPEPROTO]	クライアントでtpsubscribe()を実行した場合に、ctlがNULLでない場合や、サーバーで実行する場合に、ctlがNULLである場合です
[TPETIME]	タイムアウトが発生した場合です

- 関連関数

tpsetunsol(), tpsetunsol_flag(), tpgetunsol(), tpacall(), tpenq()

3.2.2. tpsetfd

ソケットFDをUCSプロセスの外部ソケット・スケジューラーに登録する関数です。UCS方式プロセスを使用したソケットFDを起動する際に使用されます。UCSスケジューラーは、TMM、CLHだけでなく、該当ソケットFDに到着したメッセージも一緒にチェックします。ユーザーが指定したソケットにメッセージが到着した場合、tpschedule()は特に処理をすることなく、正常結果(UCS_USER_MSG)を返します。また、どのソケットにメッセージが到着したかを確認するためには、tpisetfd()を使用します。

- プロトタイプ

```
#include <ucs.h>
int tpsetfd (int fd)
```

- パラメータ

パラメータ	説明
fd	登録するソケットFDを設定します

- 戻り値

戻り値	説明
1	関数の呼び出しに成功した場合です

戻り値	説明
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

tpsetfd()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です

- 例

```
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <errno.h>
#include <usrinc/ucs.h>
...
#define SERV_ADDR "168.126.185.129"
#define SERV_PORT 1500

int fd_read(int, char *, int);
extern int errno;

int usermain(int argc, char *argv[])
{
    ...
    int listen_fd, n, newfd;
    struct sockaddr_in my_addr, child_addr;
    socklen_t child_len;
    buf = tpalloc("STRING", NULL, 0);
    if (buf == NULL){
        error processing
    }

    memset((void *)&my_addr, NULL, sizeof(my_addr));
    memset((void *)&child_addr, NULL, sizeof(child_addr));
    listen_fd = socket(AF_INET, SOCK_STREAM, 0);
    if (listen_fd == -1){
        error processing
    }

    my_addr.sin_family = AF_INET;
```

```

inaddr = inet_addr(SERV_ADDR);
my_addr.sin_port = htons((unsigned short)SERV_PORT);

if (inaddr != -1){
    memcpy((char *)&my_addr.sin_addr, (char *)&inaddr, sizeof(inaddr));
}
ret = bind(listen_fd, (struct sockaddr *)&my_addr, sizeof(my_addr));
if (ret == -1){
    error processing
}
ret = listen(listen_fd, 5);
if (ret == -1){
    error processing
}

ret = tpsetfd(listen_fd);
if (ret == -1){
    error processing
}
...

while(1) {
    n = tpschedule(10);
    ...
    if (n == UCS_USER_MSG){
        if (tpissetfd(listen_fd)) {
            child_len = sizeof(child_addr);
            newfd = accept(listen_fd, &child_addr, &child_len);
            if (newfd == -1){
                error processing
            }

            ret = tpsetfd(newfd);
            if (ret == -1){
                error processing
            }
        }

        if (tpissetfd(newfd)){
            /* ソケットからバッファーを読み込みます */
            fd_read(newfd, buf, 1024);
            ret = tpcall("SERVICE", (char *)buf, sizeof(buf), (char **)&buf,
                        (long *)&rlen, TPNOFLAGS);
            if (ret == -1){
                error processing
            }
        }
        ...
    }
}

```

```

        tpclrfd(newfd);
        close(newfd);
    }
    ...
}
return 1;
}

```

- 関連関数

tpclrfd(), tpissetfd()

3.2.3. tpsubscribe

サーバーとクライアントで使用され、特定イベントの発生時にクライアントやサーバーからのデータ転送を受信するための要求を登録する関数です。eventnameに該当するイベントが発生した場合に通知するようTmaxシステムに登録します。

- プロトタイプ

```

#include <tmaxapi.h>
long tpsubscribe(char *eventname, char *filter, TPEVCTL *ctl, long flags)

```

- パラメータ

パラメータ	説明
eventname	NULLで終わる63文字以内の文字列で、メッセージを受信するイベントの名前を指定します。ワイルドカードやpartial-matchingなどはサポートされず、全体文字列が一致する名前のイベントのみ登録できます
filter	今後の拡張のために予約をしておいたものです。NULLを指定します
ctl	イベントが発生した場合、メッセージを取得する方式を指定する構造体です。tpsubscribe()の主体によって異なる動作をします。クライアントでtpsubscribe()を使用した場合、ctlは常にNULLである必要があり、メッセージはクライアントにunsolicitedデータ形式で渡されます。クライアントはtpsubscribe()あるいはtpgetunsol()などの関数を使用して渡されたデータを処理します
flags	現在はTPNOTIMEのみ意味があります

サーバーでtpsubscribe()を実行する場合、ctlがNULLになってはいけません。下記内容を参照してください。

```

struct tpevctl {
    long ctl_flags;
    long post_flags;
}

```

```

char svc[XATMI_SERVICE_NAME_LENGTH];
char qname[RQ_NAME_LENGTH];
};
typedef struct tpevctl TPEVCTL;

```

メンバー	説明
ctl_flags	今後の拡張のために作成されたもので、現在は0に設定する必要があります
post_flags	今後の拡張のために作成されたもので、現在は0に設定する必要があります
qname	qnameを使用する場合、メッセージはtpenq(qname、NULL、data、len、TPNOFLAGS)を使用してRQに保存されます
svc	svcを使用する場合、メッセージはtpacall(svc、data、len、TPNOREPLY)と似た方式でサーバーに渡され、サーバーの戻り値は無視されます。qnameとsvcのいずれか1つだけを使用します

- 戻り値

戻り値	説明
descriptor	関数の呼び出しに成功した場合です。tpunsubscribe()を呼び出すときに使用されるdescriptorを返します
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

tpsubscribe()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。クライアントでは、ネットワーク・エラーが最も多いです
[TPEOS]	OSレベルで発生するエラーです。メモリーの割り当てなどの問題を含みます
[TPEINVAL]	設定された引数が正しくない場合です
[TPENOENT]	TPEVCTL構造体のqnameやsvcに該当するRQやサーバーが存在しない場合です
[TPEPROTO]	クライアントとサーバーでctlがNULLでない場合です
[TPETIME]	タイムアウトが発生した場合です

- 関連関数

tpsetunsol(), tpsetunsol_flag(), tpgetunsol(), tpacall(), tpenq()

3.2.4. tpunsubscribe

サーバーとクライアントで使用され、**tpsubscribe()**で登録した特定イベントに対する要求を解除する関数です。登録されたすべての要求が解除された場合、Tmaxシステムは該当イベントの表を削除します。

- プロトタイプ

```
# include <tmaxapi.h>
int tpunsubscribe(long sd, long flags)
```

- パラメータ

パラメータ	説明
sd	tpsubscribe()で登録時に受け取った戻り値です
flags	現在はTPNOTIMEのみ意味があります

- 戻り値

戻り値	説明
正数	関数の呼び出しに成功した場合です
負数	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

tpunsubscribe()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。クライアントでは、ネットワーク・エラーが最も多いです
[TPEOS]	OSレベルのエラーが発生した場合です。メモリの割り当てなどの問題を含みます
[TPEINVAL]	設定された引数が正しくない場合です
[TPETIME]	タイムアウトが発生した場合です

- 関連関数

tpsetunsol(), tpsetunsol_flag(), tpgetunsol(), tpacall(), tpenq()

3.3. サーバー関数

3.3.1. tpclrfd

UCS方式プロセス内部のfdsetのソケットFDをオフにするのに使用される関数です。UCS方式サーバー・プロセスの外部ソケットをスケジューリングする場合に使用します。

- プロトタイプ

```
#include <ucs.h>
int tpclrfd (int fd)
```

- パラメータ

パラメータ	説明
fd	オフにする内部fdsetのソケットです

- 戻り値

戻り値	説明
1	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tpernoにエラーコードが設定されます

- エラー

tpclrfd()が正常に実行されなかった場合、tpernoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です

- 例

```
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <errno.h>
#include <usrinc/ucs.h>
...
#define SERV_ADDR "168.126.185.129"
#define SERV_PORT 1500
```

```

int fd_read(int, char *, int);
extern int errno;

int usermain(int argc, char *argv[])
{
    ...
    int listen_fd, n, newfd;
    struct sockaddr_in my_addr, child_addr;
    socklen_t child_len;
    buf = tpalloc("STRING", NULL, 0);
    if (buf == NULL){ error processing }

    memset((void *)&my_addr, NULL, sizeof(my_addr));
    memset((void *)&child_addr, NULL, sizeof(child_addr));
    listen_fd = socket(AF_INET, SOCK_STREAM, 0);
    if (listen_fd == -1){ error processing }
    my_addr.sin_family = AF_INET;
    inaddr = inet_addr(SERV_ADDR);
    my_addr.sin_port = htons((unsigned short)SERV_PORT);
    if (inaddr != -1){
        memcpy((char *)&my_addr.sin_addr, (char *)&inaddr, sizeof(inaddr));
    }

    ret = bind(listen_fd, (struct sockaddr *)&my_addr, sizeof(my_addr));
    if (ret == -1){ error processing }
    ret = listen(listen_fd, 5);
    if (ret == -1){ error processing }

    tpsetfd(listen_fd);
    ...
    while(1) {
        n = tpschedule(10);
        ...
        if (n == UCS_USER_MSG){
            if (tpissetfd(listen_fd)) {
                child_len = sizeof(child_addr);
                newfd = accept(listen_fd, &child_addr, &child_len);
                if (newfd == -1){ error processing }
                tpsetfd(newfd);
            }

            if (tpissetfd(newfd)){
                /* ソケットからバッファを読み込みます */
                fd_read(newfd, buf, 1024);
                ret = tpcall("SERVICE", (char *)buf, sizeof(buf), (char **)&buf,
                           (long *)&rlen, TPNOFLAGS);
            }
        }
    }
}

```



```

        if (ret == -1){ error processing }
        ...
        ret = tpclrfd(newfd);
        if (ret == -1){ error processing }
        close(newfd);
    }
    ...
}
return 1;
}

```

- 関連関数

tpissetfd()

3.3.2. tpissetfd

UCSプロセスでソケットFDにデータが到着したかどうかをチェックする関数です。UCS方式サーバー・プロセスの外部ソケットのスケジューリングに使用されます。

- プロトタイプ

```

#include <ucs.h>
int tpissetfd (int fd)

```

- パラメータ

パラメータ	説明
fd	テストするfdset内部のFDを設定します

- 戻り値

戻り値	説明
正数	メッセージが到着した場合です
0	メッセージが到着しなかった場合です
-1	関数の呼び出しに失敗した場合です。tpernoにエラーコードが設定されます

- エラー

tpissetfd()が正常に実行されなかった場合、tpernoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です

- 例

```
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <errno.h>
#include <usrinc/ucs.h>
...

#define SERV_ADDR "168.126.185.129"
#define SERV_PORT 1500

int fd_read(int, char *, int);
extern int errno;

int usermain(int argc, char *argv[])
{
    ...
    int listen_fd, n, newfd;
    struct sockaddr_in my_addr, child_addr;
    socklen_t child_len;

    buf = tpalloc("STRING", NULL, 0);
    if (buf == NULL){ error processing }

    memset((void *)&my_addr, NULL, sizeof(my_addr));
    memset((void *)&child_addr, NULL, sizeof(child_addr));

    listen_fd = socket(AF_INET, SOCK_STREAM, 0);
    if (listen_fd == -1){ error processing }

    my_addr.sin_family = AF_INET;
    inaddr = inet_addr(SERV_ADDR);
    my_addr.sin_port = htons((unsigned short)SERV_PORT);
    if (inaddr != -1)
        memcpy((char *)&my_addr.sin_addr, (char *)&inaddr, sizeof(inaddr));

    ret = bind(listen_fd, (struct sockaddr *)&my_addr, sizeof(my_addr));
    if (ret == -1){ error processing }
    ret = listen(listen_fd, 5);
    if (ret == -1){ error processing }
```

```

    tpsetfd(listen_fd);
    ...

    while(1) {
        n = tpschedule(10);
        ...
        if (n == UCS_USER_MSG){
            if (tpissetfd(listen_fd)) {
                child_len = sizeof(child_addr);
                newfd = accept(listen_fd, &child_addr, &child_len);
                if (newfd == -1){ error processing }
                tpsetfd(newfd);
            }

            if (tpissetfd(newfd)){
                /*ソケットからバッファを読み込みます*/
                fd_read(newfd, buf, 1024);
                ret = tpcall("SERVICE", (char *)buf, sizeof(buf), (char **)&buf,

                                (long *)&rlen, TPNOFLAGS);
                if (ret == -1){ error processing }
                ...
                tpclrfd(newfd);
                close(newfd);
            }
            ...
        }
    }
    return 1;
}

```

- 関連関数

tpissetfd(), tpsetfd()

3.3.3. tpregcb

サーバーでUCSの非同期型の要求に対する応答を受けるルーチンを設定する関数です。UCS方式プロセスがサーバー・プログラムから応答を受けたときに、それを処理するルーチンを設定します。UCS方式サーバー・プロセスに、**tpgetreply()**の代わりに使用されます。

- プロトタイプ

```
# include <ucs.h>
int  tpregcb (UcsCallback)
```

● パラメータ

パラメータ	説明
UcsCallback	UCSで非同期型要求に対する応答を処理するコールバック関数を指定します

● 戻り値

戻り値	説明
1	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tpernoにエラーコードが設定されます

● エラー

tpregcb()が正常に実行されなかった場合、tpernoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です

● 例

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

void reply_receive(UCSMSGINFO *reply);
DUMMY(TPSVCINFO *msg)
{
    data process ....
}

int usermain(int argc, char *argv[])
{
    int ret;
    char *buf

    ret = tpregcb(reply_receive);
    if (ret == -1){ error processing }
    buf=tpalloc("STRING", NULL, 0);
```

```

    if (buf==NULL) { error processing }
    data process...
    while(1)
    {
        ...
        tpschedule(3);
        cd = tpacall("SERVICE", buf, strlen(buf), TPNOFLAGS);
        if (cd < 0) { error processing }
        ...
    }
}

void reply_receive(UCSMSGINFO *reply)
{
    printf("data....%s\n", reply->data);
}

```

- 関連関数

tpunregcb()

3.3.4. tprelay

UCS形式のサーバー・プロセスでのみ使用できる関数です。サービスを要求したクライアントの情報を保存し、他のサービスを要求する方式で、マルチノード間でもサービスが行われます。この形式をとった場合、tprelay()を使用して呼び出されたサービスは、クライアントが自身を直接呼び出したものと認識します。また、サービスの実行結果は、最初にサービスを要求したクライアントに渡されます。

サービスの実行結果を呼び出し元のクライアントに渡すことができるため、UCSプロセスで簡単な構成を通じて、速い応答を誘導できます。通常、結果を取得するまでに2～3回サービスの呼び出しを行わなければならないプログラム・ルーチンの対外機関業務と連動してサービス进行处理する場合に使用するのが望ましいです。

もしtpsavctx()またはtpgetctx()によりクライアント情報を保存した後、tprelay()により他のサービスを要求していない状態でサーバー・プロセスが終了する場合には、自動でサービス呼び出し元にエラー応答が返されます。エラー応答の返却に関連しては、環境設定のSERVERセクションのCTX_EREPYオプションを参照してください。このような動作は、Tmax v5.0 SP2以降バージョンからサポートされます。

- プロトタイプ

```

#include <ucs.h>
int tprelay(char *svc, char *data, long len, long flags, CTX_T *ctxp);

```

- パラメータ

パラメータ	説明
svc	Tmax環境ファイルに登録されたサービス名を指定します
data	サービスの呼び出し時に渡されるデータです。NULLでない場合、必ずtpalloc()で割り当てられたバッファを使用します
len	送信するデータ長を指定します。CARRAY、X_OCTET、構造体の配列タイプの場合は必ず設定します
flags	現在サポートしていないパラメータで、TPNOFLAGSを設定します
ctxp	tpgetctx()あるいはtpsavectx()で取得した情報の構造体です

- 戻り値

戻り値	説明
1	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tpernoにエラーコードが設定されます

- エラー

tprelay()が正常に実行されなかった場合、tpernoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効でない場合です。たとえば、ctxpがNULLの場合、あるいは使用したバッファが正しくない場合に発生します
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です

- 例

```
...
#include <stdio.h>
#include <usrinc/ucs.h>
CTX_T *ctx = NULL;

DUMMY(TPSVCINFO *msg)
{
    data process ...
}

usermain(int argc, char *argv[])
{
    int ret, i;
    char *rcvbuf, *sndbuf;
    long sndlen;
```

```

rcvbuf = (char *)tpalloc("CARRAY",NULL, 1024);
if (rcvbuf == NULL){ error processing }
i = 0;

while(1) {
    tpschedule(1);
    if (ctx != NULL)
    {
        i++;
        if ((sndbuf = (char *)tpalloc("CARRAY",NULL, 1024)) == NULL)
        { error processing }
        else
        {
            ...
            ret = tprelay("TPRETURN", sndbuf, sndlen, 0, ctx);
            if (ret== -1) { error processing }
            data process...
            ctx = NULL;
            tpfree(sndbuf);
        }
    }
}

int RELAY(TPSVCINFO *rqst)
{
    ...
    ctx = tpsavectx();
    tpreturn(TPSUCCESS, 0, rqst->data, rqst->len, 0);
}

```

- 関連関数

tpreturn(), tpforward()

3.3.5. tpsavectx

UCSプロセスで使用され、クライアントの情報を内部的に管理する関数です。tpsavectx()はtprelay()関数と一緒に使用されます。tprelay()は、処理結果を他のサービスに渡す役割をします。一般的なサービス・プログラムでtpforward形式で別のサービスを呼び出したのと同じように動作します。結果的に呼び出されたサービスでは、処理された結果値を該当クライアントに渡します。

tpsavectx()関数は対外機関業務と同様に他のプロトコルが利用され、時間が掛かるためにチャンネルが結合される可能性が多い場合に使用されます。

一般的に使用される形式は以下のとおりです。

```
クライアント → svc1 → svc2(service, tpsavctx) → 対外機関
クライアント ← svc3 ← svc2(usermain, tprelay) ← 対外機関
```

1. クライアントがsvc1にサービスを要求します。
2. svc1はtpforward(...TPNOREPLY)を使用してsvc2を呼び出します。
3. svc2はUCSプロセスに存在するサービスで、サービス・ルーチン内でtpsavctx()を使用してクライアントの情報を保存し、対外機関と通信します。
4. 結果はusermainで受信し、tprelay()を使用してsvc3に渡します。svc3は、svc2でtpforwardを使用して自身を呼び出したものと見なし、最終結果をクライアントに渡します。

svc1でTPNOREPLYを使用してサービスを渡しているため、チャンネルが結合されるのを防ぐことができ、少数の業務プロセスでも多くのクライアントを処理できます。また、1つのUCSプロセスとして送受信プロセスの役割を兼ねることで、比較的単純なシステムを構成できます。これはシステム管理面においても効率的です。

● プロトタイプ

```
#include <ucs.h>
CTX_T * tpsavctx(void)
```

● 戻り値

戻り値	説明
CTX_T	関数の呼び出しに成功した場合です
NULL	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

● エラー

tpsavctx()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEPROTO]	tpsavctx()関数は、必ずサービス・ルーチン内で使用します。サービス・ルーチンでない位置で使用された場合、TPEPROTOエラーが発生します。 tpsvrinit() あるいは tpsvrdone() では使用できません
[TPESYSTEM]	メモリーの割り当てエラーが発生した場合です

● 例


```

...
#include <stdio.h>
#include <usrinc/ucs.h>

CTX_T *ctx = NULL;

usermain(int argc, char *argv[])
{
    int ret, i;
    char *rcvbuf, *sndbuf;
    long sndlen;

    rcvbuf = (char *)tpalloc("CARRAY",NULL, 1024);
    if (rcvbuf == NULL){ error processing }

    i = 0;

    while(1) {
        tpschedule(1);
        if (ctx != NULL)
        {
            i++;
            if ((sndbuf = (char *)tpalloc("CARRAY",NULL, 1024)) == NULL)
            { error processing }
            else
            {
                ...
                ret = tprelay("TPRETURN", sndbuf, sndlen, 0, ctx);
                if (ret== -1) { error processing }
                data process...
                ctx = NULL;
                tpfree(sndbuf);
            }
        }
    }
}

int RELAY(TPSVCINFO *rqst)
{
    ...
    ctx = tpsavctx();
    tpreturn(TPSUCCESS, 0, rqst->data, rqst->len, 0);
}

```

- 関連関数

tpreturn(), tpforward(), tprelay()

3.3.6. tpschedule

UCS形式のサーバー・プロセスでのみ使用される関数で、UCSサーバー・プロセスでデータの到着を待機します。最大タイムアウト時間の間待機し、その間にデータが到着した場合は即時に返します。

tpschedule()関数は、データの到着時に該当するサービスが自動的に実行された後に返されます。そのため、データが到着後にユーザーが任意でサービスを実行してはいけません。

注

サービスは常にシステムによって実行されるため、UCS形式のサービス・プログラムでもこの点は注意してください。

● プロトタイプ

```
#include <ucs.h>
int tpschedule(int timeout)
```

● パラメータ

パラメータ	説明
timeout	待機する時間を秒単位で入力します – -1: データが到着したかどうかをチェックのみ行った後、すぐに返します – 0: データが到着するまで無限に待機します

● 戻り値

戻り値	説明
正の整数	関数の実行に成功してデータが到着した場合です
-1	タイムアウト時間までにデータが到着していない場合です。あるいは、関数が実行に失敗してエラーが発生した場合です。タイムアウト時間までデータが到着しなかった場合は-1を返し、tperrnoに13番(TPETIME)が設定されます。それ以外の場合、tperrnoにエラーコードが設定されます

● エラー

tpschedule()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます

エラーコード	説明
[TPEOS]	運用システムにエラーが発生した場合です
[TPETIME]	タイムアウト時間までにデータが到着していない場合です

- 例

```
#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/ucs.h>
int usermain(int argc, char *argv[])
{
    ...
    while(1)
    {
        ...
        tpschedule(3);
        ret = tpcall("SERVICE", (char *)buf, strlen(buf), (char **)&buf,
                    (long *)&rlen, TPNOFLAGS);
        if (ret == -1) { error processing}
        ...
    }
}
```

- 関連関数

tpsleep(), tp_sleep(), tp_usleep()

3.3.7. tpsendtocli

サーバーで使用される関数で、指定されたクライアントに非要求メッセージを送信します。**tpbroadcast()**は、Tmaxシステムに接続されている任意のクライアントに非要求メッセージを送信します。**tpsendtocli()**は、サーバー・プロセスで該当サーバー・プロセスが提供するサービスを要求したクライアントにのみ非要求メッセージを送るために使用します。

- プロトタイプ

```
# include <tmaxapi.h>
int tpsendtocli (int clid, char *data, long len, long flags)
```

- パラメータ

パラメータ	説明
clid	tpgetclid()で取得したクライアントの一意の番号です

パラメータ	説明
data	tpalloc()によって割り当てられたバッファです。dataが指すバッファが、長さを指定する必要がないタイプの場合、lenは無視され、一般的に0が使用されます。dataが指すバッファが、長さの指定が必要なタイプの場合、lenは0になれません。dataがNULLの場合、lenは無視されます
len	送信するバッファ長です
flags	<p>flagsで使用可能な値は以下のとおりです</p> <ul style="list-style-type: none"> – TPNOFLAG(0) <p>メッセージはクライアントが受信します。しかし、クライアントが受信したメッセージを早急に処理できなければ、要求した結果の受信に時間が掛かることがあります</p> <ul style="list-style-type: none"> – TPUDP <p>TPUDPフラグは、クライアントとデータを通信する方式がUDPという意味ではありません。呼び出し元がデータを送信する際、送信する内部バッファに渡されるメッセージがいっぱいになって送信できない場合があります。この場合、データは捨てても構わないという意味です。通信のUDPのように、データが途中で紛失することがあります</p> <ul style="list-style-type: none"> – TPFLOWCONTROL <p>クライアントの状態をチェックし、他のメッセージを要求できるかどうかを確認します。該当クライアントの送信メッセージが多く蓄積している場合、tpsendtcli()は-1を返し、tperrnoをTPEQFULLに設定します。このフラグは、Tmaxシステムの負荷を減らします</p>

- 戻り値

戻り値	説明
1	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

tpsendcli()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEBADDESC]	clidが有効でない場合です
[TPEPROTO]	tpsendtcli()が不適切な状況で呼び出された場合です

エラーコード	説明
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です
[TPEQFULL]	送信メッセージがあるため、同じメッセージである場合は再送信する必要がありません

- 例

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

SERVICE(TPSVCINFO *msg)
{
    int ret, clid;
    char *buf;

    buf = (char *)tpalloc("STRING", NULL, 0);
    if (buf==NULL) { error processing }
    strcpy(buf, msg->data);
    data process....
    clid = tpgetclid();
    if (clid== -1) { error processing }

    ret=tpsendtocli(clid, (char *)buf, 0, 0);
    if (ret== -1) { error processing }
    tpreturn(TPSUCCESS, 0, 0, 0);
}
```

- 関連関数

tpbroadcast()

3.3.8. tpsvctimeout

サービス・タイムアウトが発生した場合に呼び出されるルーチンです。サービス・タイムアウトが発生した場合、サーバープログラムは自動的にtpsvctimeout()を呼び出します。ユーザーが関数を再定義した場合、再定義した関数を呼び出します。

サービスを実行中にtx_commit()、tx_rollback()関数を実行する場合、内部でxa_commit()、xa_rollback()段階でサービス・タイムアウトが発生すると、それを区別するためにtperrnoにTPETRANを設定して、tpsvctimeout関数内で参照できるようにします。

- プロトタイプ

```
# include <tmaxapi.h>
void tpsvctimeout(TPSVCINFO *msg)
```

- パラメータ

パラメータ	説明
msg	タイムアウトが発生したサービスの呼び出し時に使用したメッセージです

- 戻り値

tpsvctimeout()は、サービスのタイムアウトが発生した場合、開発者が必要な作業を実行するように作成する関数です。戻り値はなく、エラーも発生しません。

- 例

```
#include <stdio.h>
#include <usrinc/atmi.h>
SERVICE(TPSVCINFO *msg)
{
    ...
    tpreturn(TPSUCCESS, 0, (char *)msg->data, 0, TPNOFLAGS);
}

void tpsvctimeout(TPSVCINFO *msg)
{
    ...
    tpreturn(TPFAIL, TPETIME, (char *)buf, sizeof(buf), TPNOFLAGS);
}
```

3.3.9. tpsvrdone

UCS方式サーバー・プロセスを終了する関数です。Tmaxアプリケーション・サーバー・プログラムの分離されたmainは、サービス要求処理をすべて終え、プロセスを終了する前にtpsvrdone()を呼び出します。ルーチンの実行時、そのサーバー・プロセスはシステムの一部ではありますが、サービスはサポートされません。tpsvrdone()ルーチン内でTmax通信が実行されたり、トランザクションが定義されることもあります。

tpsvrdone()が対話型接続を維持している場合は、Tmaxは対話型接続を終了します。非同期性応答を待機している場合は、待機していた非同期性応答を無視します。トランザクション・モードの間はトランザクションを停止し、サーバーはすぐに終了されます。

アプリケーションでtpsvrdone()ルーチンを提供しない場合、Tmaxが提供するデフォルト・ルーチンが代わりに呼び出されます。デフォルトのtpsvrdone()は、トランザクションを処理するサーバー・グループに含まれた

サーバーであれば、**tx_close()**と**userlog()**を呼び出して、サーバーがすぐに終了することを通知します。**tpreturn()**と**tpforward()**のうち1つが**tpsvrdone()**内で呼び出された場合、このようなルーチンは何も作動せず、返すだけです。

- プロトタイプ

```
# include <tmaxapi.h>
int tpsvrdone(void)
```

- 戻り値

tpsvrdone()は、開発者がサーバー・プロセスの終了を実行する前に必要な作業を実行するように作成する関数です。戻り値はなく、エラーも発生しません。

- 例

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

EXEC SQL INCLUDE sqlca.h;

SERVICE(TPSVCINFO *msg)
{
    int ret, cd;
    char *buf;

    EXEC SQL begin declare section;
    ...
    EXEC SQL end declare section;
    EXEC SQL CONNECT : scott IDENTIFIED BY : tiger;
    buf=tpalloc("STRING", NULL, 0);
    if (buf==NULL) { error processing }
    data process....

    cd=tpgetclid();
    if (cd==-1) { error processing }
    ret=tpsendtocli(cd, buf, 0, TPNOFLAGS);
    if (ret==-1) { error processing }
    data process....

    tpsvrdone();
}

void tpsvrdone()
{
    printf(" Sevice end\n");
}
```

```
EXEC SQL COMMIT WORK RELEASE;
}
```

- 関連関数

tx_close(), tpsvrinit()

3.3.10. tpsvrdown

UCS方式サーバー・プロセスを正常終了させる関数です。UCS方式サーバー・プロセスは、一般的に外部システムとの通信に多く使用されます。代表的な例は、銀行システム間の通信です。何らかの問題が発生し、外部システムがサービス要求を処理できないような状況を避ける方法が必要です。

tpsvrdown()は、UCS方式のサーバー・プロセスを正常終了させるため、障害が発生した外部システムにそれ以上サービスを要求しません。したがって、リソースを節約できるだけでなく、外部システムの負荷も減らすことができます。

- プロトタイプ

```
#include <ucs.h>
int tpsvrdown(void)
```

- 戻り値

tpsvrdown()は、開発者がサーバー・プロセスの終了を実行する前に必要な作業を実行する関数です。そのため、戻り値がなく、エラーも発生しません。

- 例

```
...
#include <usrinc/atmi.h>
#include <usrinc/ucs.h>
int usermain(int argc, char *argv[])
{
    count=0;
    ...
    while(1)
    {
        ...
        tpschedule(3);
        cd = tpacall("SERVICE", buf, strlen(buf), TPNOREPLY);
        if (cd < 0) { error processing }
        ...
        if (count == 10) tpsvrdown();
    }
}
```



```

}

void reply_receive(UCSMMSGINFO *reply)
{
    printf("data....%s\n", reply->data);
}

```

- 関連関数

tpsvrinit(), tpsvrdone()

3.3.11. tpunregcb

UCS方式のサーバー・プロセスから非同期型の要求に対する応答を受けるルーチンを再設定する関数です。サーバー・プログラムから応答を受けた際に実行されるルーチンを再設定(reset)します。

- プロトタイプ

```

#include <ucs.h>
int tpunregcb (void)

```

- 戻り値

戻り値	説明
1	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

tpunregcb()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です

- 例

```

...
#include <usrinc/atmi.h>
#include <usrinc/ucs.h>
void reply_receive(UCSMMSGINFO *reply);

```

```

int usermain(int argc, char *argv[])
{
    ...
    ret = tpregcb(reply_receive);
    if (ret == -1){ error processing }

    ret = tpunregcb();
    if (ret == -1){ error processing }
    while(1)
    {
        ...
        tpschedule(3);
        cd = tpacall("SERVICE", buf, strlen(buf), TPNOFLAGS);
        if (cd < 0) { error processing }
        ...
    }
}

void reply_receive(UCSMSGINFO *reply)
{
    printf("first reply receive\n");
    printf("data....%s\n", reply->data);
}

```

- 関連関数

tpregcb()

3.4. クライアント関数

3.4.1. tpgetunsol

クライアントの要求なく、一方的に渡されたメッセージを処理する関数です。メッセージを送る側で、**tpbroadcast()**あるいは**tpsendtoci()**、**tppost()**を使用して渡します。

tpgetunsol()の呼び出し前に渡された一方的なメッセージは無視されます。tpgetunsol()を使用して非要求メッセージを受け取るには、tpstart()を使用し、Tmaxシステムの接続時にflagsをTPUNSOL_POLLあるいはTPUNSOL_HNDと指定する必要があります。tpgetunsol()がプログラムに呼び出された場合、tpstart()のフラグがTPUNSOL_IGNと設定されていても、内部的にTPUNSOL_POLLに転換され、サーバーから非要求メッセージを受け取ります。

- プロトタイプ

```
#include <tmxapi.h>
int  tpgetunsol (int type, char **data, long *len, long flags)
```

● パラメータ

パラメータ	説明
type	サーバーから渡されたメッセージの形式です。UNSOL_TPPOST、UNSOL_TPROADCAST、UNSOL_TPSENDTOCLIなどがあります
data	渡されたメッセージについてのポインターです。クライアントが分からないバッファ・タイプおよびサブタイプの場合があり得ますが、この場合dataを使用できません
len	メッセージの長さの合計です
flags	<p>メッセージのブロッキング処理可否を決定します。</p> <p>flagsで使用可能な値は以下のとおりです</p> <ul style="list-style-type: none"> – TPBLOCK <p>tpgetunsol()の呼び出し時にブロッキング状態で一方的なメッセージが来るまで待機します</p> – TPNOCHANGE <p>受信した応答バッファと*dataが指すバッファのタイプが異なる場合、受信者が認識できる限度内で、*dataのバッファ・タイプは受信した応答バッファ・タイプに変更されます。フラグが設定された場合、*dataが指すバッファのタイプは変更できません。受信した応答バッファのタイプおよびサブタイプは、*dataが指すバッファのタイプおよびサブタイプと一致する必要があります</p> – TPNOTIME <p>関数の呼び出し元がブロック・タイムアウトを無視し、応答が受信されるまで無限に待機します。トランザクション・モードでtpgetrply()を実行した場合、トランザクション・タイムアウトが適用されます</p> – TPSIGRSTRT <p>シグナルの割り込みを受け入れる場合に使用します。システム関数の呼び出しが妨害されたとき、システム関数の呼び出しが再実行されます。TPSIGRSTRTフラグが設定されていない状態でシグナルの割り込みが発生した場合、関数は失敗し、tperrnoにTPGOTSIGが設定されます</p> – TPGETANY <p>入力値の記述子(cd)を無視し、これに関係なく受信可能な応答を返します。cdは返された応答に対する呼び出し記述子になります。応答がなければ、一般的にtpgetrply()は応答が到着するまで待機します</p>

- 戻り値

戻り値	説明
1	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tperrnoに状況に該当する値が設定されます

- エラー

tpgetunsol()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEPROTO]	tpgetunsol()が不適切な状況で呼び出された場合です。たとえば、サーバー内で呼び出された場合に発生します
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です

- 例

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

void main(int argc, char *argv[])
{
    int ret;
    char *buf;
    long len;

    ret=tpstart((TPSTART_T *)NULL);
    if (ret==-1) { error processing }

    buf=tpalloc("STRING", NULL, 0);
    if (buf==NULL) { error processing }
    data process....
    while(1)
    {
        ret=tp_sleep(2);
        if (ret==-1) { error processing }
        if (ret==0) printf("nothing happened\n");
        else {
            ret=tpgetunsol(UNSOL_TPSENDTOCLI, (char **)&buf, &len, TPNOCHANGE);

            if (ret==-1) { error processing }
```

```

        printf("received data : %s\n", buf);
    }

    data process....
    if (strncmp(buf, "end", 3)==0) break;
}

data process....
tpfree(buf);
tpend();
}

```

- 関連関数

tpbroadcast(), tpsetunsol(), tpstart(), tpend()

3.4.2. tpsetunsol

クライアントで使用され、非要求受信メッセージを処理するルーチンを設定する関数です。tpsetunsol()が初めて呼び出される前にTmaxライブラリーによって受信された非要求メッセージは無視されます。NULL関数ポインターのtpsetunsol()の呼び出しも同様に無視されます。

システムが非要求メッセージの通知を受けるのに使用される方法は、アプリケーションに任意で決定されます。これは各クライアント別に変更が可能です。呼び出しで渡された関数ポインターは、与えられたパラメータ定義に適合する必要があります。

- プロトタイプ

```

# include <atmi.h>
Unsolfunc *tpsetunsol (void ( *disp ) ( char *data, long len, long flags )
)

```

- パラメータ

パラメータ	説明
data	受信された型付きバッファを指示します。データが伴っていない場合、dataはNULLになれません。dataが、クライアントが認識できないバッファ・タイプおよびサブタイプである場合、データは理解されません。アプリケーションはdataを削除できず、代わりにシステムがこれを削除し、データ領域を無効化して返します
len	データの長さを表します
flags	現在は使用されていません

- 戻り値

戻り値	説明
ポインター/NULL	関数の呼び出しに成功した場合です – ポインター: 以前設定された非要求メッセージ処理ルーチンのポインターを返します – NULL: 以前にいかなるメッセージ処理関数も設定されなかったことを表します。正常なリターンです
TPUNSOLERR	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

● エラー

tpsetunsol()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEPROTO]	tpsetunsol()が不適切な状況で呼び出された場合です
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されません
[TPEOS]	運用システムにエラーが発生した場合です

● 例

```
#include <stdio.h>
#include <usrinc/atmi.h>
#include "../sdl/demo.s"

void get_unsol(char *data, long len, long flags)
{
    printf("get unsolicited data = %s\n", data);
    data process....
}

void main(int argc, char *argv[])
{
    int ret;
    char *buf;
    long len;

    ret=tpstart((TPSTART_T *)NULL);
    if (ret==-1) { error processing }

    ret=tpsetupsol_flag(TPUNSOL_HND);
    if (ret==-1) { error processing }
```

```

ret=tpsetunsol(get_unsol);
if (ret==TPUNSOLERR) { error processing }

buf = (char *)tpalloc("CARRAY", NULL, 20);
if (buf==NULL) {error processing };

data process...

ret=tpcall("SERVICE", buf, 20, (char **)&buf, &len, TPNOFLAGS);
if (ret==-1) { error processing }

data process...

tpfree((char *)buf);
tpend();
}

```

- 関連関数

tpstart(), tpend(), tpgetunsol()

3.4.3. tpsetunsol_flag

クライアントで使用され、非要求メッセージの受信フラグを変更する関数です。**tpstart()**関数を使用してTmaxシステムと接続する場合、使用したフラグ値を再設定します。

- プロトタイプ

```

# include <atmi.h>
int tpsetunsol_flag (int flag)

```

- パラメータ

パラメータ	説明
flag	<p>非要求メッセージを、受信と関連したフラグに設定します。以下のうち1つを設定します</p> <ul style="list-style-type: none"> – TPUNSOL_IGN : 非要求メッセージを受けません – TPUNSOL_HND, TPUNSOL_POLL : 非要求メッセージを受けます

- 戻り値

戻り値	説明
1	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。flagが設定可能な3つのうち1つでない場合で、エラーが発生してもtperrnoには値が設定されません

- 例

```
#include <stdio.h>
#include <usrinc/atmi.h>
#include "../sdl/demo.s"
void get_unsol(char *, long, long);
void main(int argc, char *argv[])
{
    int ret;
    char *buf;
    long len;

    ret=tpstart((TPSTART_T *)NULL);
    if (ret==-1) { error processing }

    ret=tpsetupsol_flag(TPUNSOL_HND);
    if (ret==-1) { error processing }

    ret=tpsetunsol(get_unsol);
    if (ret==TPUNSOLERR) { error processing }
    buf = (char *)tpalloc("CARRAY", NULL, 20);
    if (buf==NULL) {error processing };
    data process...
    ret=tpcall("SERVICE", buf, 20, &buf, &len, TPNOFLAGS);
    if (ret==-1) { error processing }
    data process...
    tpfree((char *)buf);
    tpend();
}

void get_unsol(char *, long, long);
{
    printf("get unsolicited data.\n");
    data process....
}
```

- 関連関数

tpstart()

第4章 UCSの使用例

UCSは、業務の特性によって多様にプログラム化して使用します。本章では、実際にUCSが使用される際に様々な方法でプログラム化された例について説明します。

4.1. プログラムの異常終了管理

以下は、異常終了プログラム(core)が発生した場合、該当するコアファイルを特定のディレクトリーに移動させ、接続されたクライアントに異常終了プログラムの情報を渡す例です。

<ucs_svr1.c>

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>
#include <sys/time.h>
#include <sys/types.h>
#include <dirent.h>
#include <fcntl.h>
#include <usrinc/atmi.h>

char _file_path[100];
int _cliid[10];

int tpsvrint(int argc, char *argv[])
{
    return 0;
}

int tpsvrdone()
{
    return 0;
}

int usermain(int argc, char *argv[])
{
    int iRet, i;
    char core_file[50];
    char server_name[50];
    char cur_time[20];
    char *str;
```

```

/* init cliid */
for (i=0; i<10; i++)
    _cliid[i] = -1;
strcpy(_file_path, argv[1]);
sprintf(core_file, "%s/core", _file_path);
while(1) {
    tpschedule(5);
    iRet = checkCoreFile();
    if (iRet == 1) {
        iRet = getCoreName(core_file, server_name);
        iRet = getCurrentTime(cur_time);
        iRet = moveCoreFile(core_file, server_name, cur_time);
        str = (char *)tpalloc("STRING", NULL, 0);
        sprintf(str, "%s program core !!", server_name);
        for (i=0; i<10; i++) {
            if (_cliid[i] < 0) continue;
            iRet = tpsendtocli(_cliid[i], str, strlen(str), 0);
            if (iRet == -1) {
                printf("client close connect !!\n");
                _cliid[i] = -1;
            }
        }
        for (i=0; i<10; i++)
            printf("cliid = %d - %d\n", i, _cliid[i]);
        tpfree(str);
    }
}

int checkCoreFile()
{
    char    server_name[50];
    char    core_file[100];
    struct  dirent *dirent;
    DIR     *dp;
    dp = opendir(_file_path);
    if (dp == NULL) {
        printf("%s directory is not found !\n", _file_path);
        return -1;
    }
    for (dirent = readdir(dp); dirent != NULL; dirent = readdir(dp)) {
        if ( (strlen(dirent->d_name) == 4) &&
            (strncmp(dirent->d_name, "core", 4) == 0)) {
            closedir(dp);
            return 1;
        }
    }
}

```

```

        closedir(dp);
        return -1;
    }

int getCoreName(char *filename, char *server)
{
    int fd, cnt, i;
    char buf[6000];
    fd = open(filename, O_RDONLY);
#ifdef _HP
    cnt = read(fd, buf, 144);
#endif
#ifdef _SUN
    cnt = read(fd, buf, 132);
#endif
#ifdef _IBM
    cnt = read(fd, buf, 1759);
#endif
    cnt = read(fd, buf, 1760);
    while(1) {
        cnt = read(fd, buf, 1);
        if (cnt != 1) return -1;
        *server++ = buf[0];
        if (buf[0] == 0) break;
    }
    close(fd);
}

int getCurrentTime(char *cur_time)
{
    struct tm *tm;
    time_t tnow;
    time(&tnow);
    tm = localtime(&tnow);
    sprintf(cur_time, "%04d%02d%02d%02d%02d",
            tm->tm_year+1900, tm->tm_mon+1, tm->tm_mday,
            tm->tm_hour,      tm->tm_min,      tm->tm_sec);
    return 0;
}

int moveCoreFile(char *core, char *server, char *time)
{
    char cmd[100];
    char file_name[50];
    printf("server = [%s]\n", server);
    sprintf(file_name, "core_%s_%s", server, time);
    sprintf(cmd, "mv %s %s/%s", core, _file_path, file_name);
}

```

```

    system(cmd);
    return 0;
}

CONN_TERM(TPSVCINFO *msg)
{
    int i;
    char *stdata;
    stdata = (char *)msg->data;
    for (i=0; i<10; i++) {
        if (_cliid[i] >= 0) continue;
        _cliid[i] = tpgetclid();
        printf("connect client %d = %d\n", i, _cliid[i]);
        break;
    }
    tpreturn(TPSUCCESS, 0, (char *)stdata, 0, 0);
}

```

4.2. ソケットを使用した非同期通信プログラム

以下は、クライアントから入ってくるソケットを受け入れ、受け入れたソケットから入ってくる要求をtpacallした後に、その結果を再度クライアントに送信する例です。

<ucs_svr2.c> usermain()

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <signal.h>
#include <sys/types.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 <usrinc/atmi.h>
#include <usrinc/ucs.h>

#ifndef INADDR_NONE
#define INADDR_NONE 0xffffffff
#endif

```

```

#define CLIENT_PORT      9345
#define HOST_ADDR        "61.33.32.107"
#define MAX_BUFFER_LEN   4096

/* ----- global variable ----- */
int      client_fd = -1;
char      ip_addr[30];
int      portno;

extern int _cur_clhfd;

/* ----- service initial ----- */
tpsvrinit(int argc, char *argv[])
{
    sleep(5);
    parse_args(argc, argv);

    client_fd = network_connect(ip_addr, portno);
    if (client_fd > 0)
        tpsetfd(client_fd);

    printf("ucs_sample:client_fd = %d\n", client_fd);
    printf("ucs_sample: _cur_clhfd(1) = %d\n", _cur_clhfd);
}

tpsvrdone()
{
    if (client_fd > 0) {
        tpclrfd(client_fd);
        close(client_fd);
    }
}

/* ----- main ----- */
int usermain(int argc, char *argv[])
{
    int    n;

    /* never return */
    while(1) {
        if (client_fd < 0) {
            client_fd = network_connect(ip_addr, portno);
            if (client_fd > 0)
                tpsetfd(client_fd);
            else {
                tpschedule(5);
            }
        }
    }
}

```

```

        continue;
    }
}

printf("ucs_sample:client_fd = %d\n", client_fd);
printf("ucs_sample: _cur_clhfd(1) = %d\n", _cur_clhfd);

    if ((n = tpschedule(0)) < 0) {
        sleep(1);
        continue;
    }

printf("ucs_sample: _cur_clhfd(2) = %d\n", _cur_clhfd);
    if (tpissetfd(client_fd)) {
        if ((n = request_from_client(client_fd)) < 0) {
            tpclrfd(client_fd);
            close(client_fd);
            client_fd = -1;
        }
    }
}

}

/* ----- command argument ----- */
int parse_args(int argc, char *argv[])
{
    int c;

    portno = -1;
    memset(ip_addr, 0x00, sizeof(ip_addr));

    opterr = 0; /* don't want getopt() writing to stderr */
    while ((c = getopt(argc, argv, "i:p:")) != EOF) {
        switch (c) {
            case 'p': /* port */
                portno = atoi(optarg);
                break;
            case 'i': /* ip-addr */
                strcpy(ip_addr, optarg);
                break;
            case '?':
                printf("unrecognized option: -%c", optopt);
        }
    }

    /* default value: portno, shared memory key */
    if (portno <= 0) {

```

```

        portno = CLIENT_PORT;
        printf("no PORT is set: assumed %d\n", portno);
    }
    if (ip_addr[0] == 0x00) {
        strcpy(ip_addr, HOST_ADDR);
        printf("no IP-ADDR is set: assumed %s\n", ip_addr);
    }
    return 1;
}

/* ----- client request ----- */
int request_from_client(int fd)
{
    int    n, len;
    char    *ptr, buffer[MAX_BUFFER_LEN];

    /* read header */
    memset(buffer, 0x00, sizeof(buffer));
    n = socket_read(fd, buffer, 4);
    if (n <= 0)
        return -1;

    len = atoi(buffer);
    printf("ucs_sample:length : %d\n", len);

    /* read data */
    n = socket_read(fd, &buffer[4], len);
    if (n <= 0) {
        return -1;
    }

    sleep(3);
    len += 4;
    n = socket_write(fd, buffer, len);

    printf("ucs_sample:socket write : n=%d\n", n);
    return n;
}

/* ----- client connect for TCP/IP ----- */
int network_connect(char *host, int port)
{
    struct sockaddr_in    serv_addr;
    unsigned long    inaddr;
    struct hostent    *hp;
    int    i, fd;

```

```

memset((char *) &serv_addr, 0, sizeof(serv_addr));
serv_addr.sin_family = AF_INET;
serv_addr.sin_port   = htons(port);

/* First try to convert the host name as a dotted-decimal number.
 * Only if that fails do we call gethostbyname(). */
if ((inaddr = inet_addr(host)) != INADDR_NONE) {
    /* it's dotted-decimal */
    memcpy((char *) &serv_addr.sin_addr, (char *) &inaddr, sizeof(inaddr));

} else {
    if ((hp = gethostbyname(host)) == NULL) {
        printf("host name error: %s\n", host);
        return(-1);
    }
    memcpy((char *) &serv_addr.sin_addr, hp->h_addr, hp->h_length);
}

if ((fd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
    printf("can't open stream socket\n");
    return -1;
}

if (connect(fd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) >= 0)
    return fd;

close(fd);
return -1;
}

/* ----- data read ----- */
int socket_read(int fd, char *ptr, int nbytes)
{
    int      nleft, nread;
    char      *ptr2;

    ptr2 = (char *)ptr;
    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 */
}

/* ----- data write ----- */
int socket_write(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);
}

```

4.3. RQを使用したプログラム

以下は、失敗キューに蓄積されたデータをデキューし、再度呼び出すプログラムの例です。

<ucs_svr3.c>

```

#include    <sys/types.h>    /* required for some of our prototypes */
#include    <stdio.h>
#include    <stdlib.h>
#include    <string.h>
#include    <unistd.h>
#include    <fcntl.h>

#include    <sys/socket.h>
#include    <sys/stat.h>
#include    <sys/un.h>
#include    <netinet/in.h>
#include    <arpa/inet.h>

```

```

#include      <usrinc/atmi.h>
#include      <usrinc/tmaxapi.h>

#define      MAX_BUF_SZ      10000
#define      SZ      100
#define      QUEFILE      "rq1"
#define      QUESERVICE      "+fail"

int usermain(int argc, char*argv[])
{
    char svc[XATMI_SERVICE_NAME_LENGTH];
    long len, n;
    int try, iFailCnt;
    char *ptr;
    char QueFile[SZ];
    char QueService[SZ];

    strcpy(QueFile, QUEFILE);
    strcpy(QueService, QUESERVICE);

    ptr = (char *)tpalloc("CARRAY", NULL, MAX_BUF_SZ);

    while (1) { /* Endless Loop */
        tpschedule(10); /* Sleep 10 seconds */

        n = 0;
        try = 0;

        iFailCnt = tpqstat(QueFile, TMAX_FAIL_QUEUE);

        while( (n >= 0) && (try++ < iFailCnt) ) {
            n = tpdeq(QueFile, QueService, &ptr, &len, TPRQS );

            if (n < 0) {
                if (tperrno == TPEMATCH) /* Fail Q empty */
                    ;
                else
                    printf("tpdeq fail[%s]\n", tpstrerror(tperrno));
                continue;
            }
            n = tpextsvcname((char*)ptr, svc);

            if (n < 0) {
                printf("tpextsvcname fail![%s]\n", tpstrerror(tperrno));
                continue;
            }
            n = tpenq(QueFile, svc, (char*)ptr, len, TPRQS);
        }
    }
}

```

```

        if (n < 0) {
            printf("tpenq fail! [%s]\n", tpstrerror(tperrno));
            continue;
        }
    }
}
return 1;
}

```

4.4. クライアント対サーバー・プログラム

以下は、サーバー・プログラムでループを実行し、クライアントでLOGINサービスを要求するとUCSが応答を送信する例です。

設定ファイル

```

*DOMAIN
tmax1          SHMKEY = 79970, MINCLH=1, MAXCLH=3,
               TPORTNO=8844, BLOCKTIME=120

*NODE
tmaxs2         TMAXDIR = "/user/jaya/tmax3511",
               APPDIR  = "/user/jaya/tmax3511/appbin",
               PATHDIR = "/user/jaya/tmax3511/path",
               TLOGDIR = "/user/jaya/tmax3511/log/tlog",
               ULOGDIR = "/user/jaya/tmax3511/log/ulog",
               SLOGDIR = "/user/jaya/tmax3511/log/slog"

*SVRGROUP
tmaxs2_nx      NODENAME = "tmaxs2"

*SERVER
ucs_server     SVGNAME = tmaxs2_nx, SVRTYPE = UCS

*SERVICE
LOGIN          SVRNAME = ucs_server

```

クライアント・プログラム

1. クライアント・プログラムを実行すると、tpstart()を行います。LOGINというサービスをtpcall()し、UCSプロセスにサービスを要求します。
2. クライアント・プログラムに設定されているtpsetunsol_flag(TPUNSOL_POLL);は、サーバー・プロセスから送るメッセージを受信するという意味です。

3. クライアント・プログラムは、サーバー・プロセスでsndbufに格納して送った「Client Registration Success」というメッセージをrcvbufから受け取り、出力します。

```
printf("After tpcall() received Message from server:%s\n", rcvbuf);
```

4. クライアントから要求があればサーバー・プロセスのloop countが増加します。countが増加しながらサーバー・プロセスのfor loopからsndbufに格納して送信した「Success tpsendtocli [0]」メッセージが、同様にクライアントでもwhile loopでloop countが増加しながら出力されます。

```
while(1) {
    tpgetunsol(UNSOL_TPSENDTOCLI, &rcvbuf, &rcvlen, TPBLOCK);
    printf("Loop Count : %d\n", RecvCnt);
    if(rcvlen > 0) {
        printf("Counter : %d #[Received Data from Server : %s]\n",
            RecvCnt, rcvbuf);
        RecvCnt ++;
    }
}
```

以下は、クライアント・プログラムの例です。

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

main(int argc, char *argv[])
{
    char *sndbuf;
    char *rcvbuf;
    long rcvlen;
    int RecvCnt = 0;

    if(tpstart((TPSTART_T *)NULL) == -1)
    {
        error processing...
    }

    tpsetunsol_flag(TPUNSOL_POLL);

    if((sndbuf = (char *)tpalloc("CARRAY", NULL, 1024)) == NULL)
    {
        error processing...
    }

    if((rcvbuf = (char *)tpalloc("CARRAY", NULL, 1024)) == NULL)
    {
```

```

        error processing...
    }

    if(tpcall("LOGIN", sndbuf, 1024, &rcvbuf, &rcvlen, 0) == -1)
    {
        error processing...
    }

    printf("After tpcall() received Message from server:%s\n", rcvbuf);

    while(1)
    {
        tpgetunsol(UNSOL_TPSENDTOCLI, &rcvbuf, &rcvlen, TPBLOCK);
        printf("Loop Count : %d\n", RecvCnt);
        if(rcvlen > 0)
        {
            printf("Counter : %d #[Received Data from Server : %s]\n",

                    RecvCnt, rcvbuf);
            RecvCnt ++;
        }
        if (RecvCnt == 10) break;
    }
    tpfree((char *)sndbuf);
    tpfree((char *)rcvbuf);
    tpend();
}

```

クライアント・プログラムの例の結果は以下のとおりです。

```

tmaxs2:/user/jaya/tmax3511/sample/client>ucs_client
After tpcall() received Message from server:Client Registration Success
Loop Count : 0
Counter : 0 #[Received Data from Server : Success tpsendtocli [0]]
Loop Count : 1
Counter : 1 #[Received Data from Server : Success tpsendtocli [1]]
Loop Count : 2
Counter : 2 #[Received Data from Server : Success tpsendtocli [2]]
Loop Count : 3
Counter : 3 #[Received Data from Server : Success tpsendtocli [3]]
Loop Count : 4
Counter : 4 #[Received Data from Server : Success tpsendtocli [4]]

```

サーバー・プログラム

1. サーバー・プログラムは、Tmaxが起動してから5秒間一時停止し、クライアントからの要求があるまで“loop execute...0”を出力して、無限ループし続けます。

```
sleep(5);
printf ("loop execute... %d\n", count);
```

無限ループをし、Whileループの最後の行にあるjobs = tpschedule(-1);部分で、クライアントから送信した要求があるか否かを確認します。要求がなければ再度ループしますが、要求があればその要求を受け取ります。

```
client_id[num_cli] = tpgetclid();
printf("client id(clid) = %d\n", client_id[num_cli]);
num_cli++;
```

2. 保存されているクライアントのID値としてWhileループのForループではカウント数が増加します。その増加した値をsndbufに格納し、tpsendtocli()関数を使用してクライアントにサービスを実行します。

```
for (i = 0; i < num_cli; i++)
{
    sprintf(sndbuf, "Success tpsendtocli [%d]", count++);
    /* クライアントIDを参照し、顧客にデータを送信する部分 */
    tpsendtocli (client_id[i], sndbuf, 1024, 0);
}
```

3. sndbufにクライアントが登録されたというメッセージを格納してtpreturn()します。

```
sprintf(sndbuf, "Client Registration Success");
tpreturn(TPSUCCESS, 0, (char *)sndbuf, 1000, 0);
```

以下は、サーバー・プログラムの例です。

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

#define MAX_CLI 100

int num_cli;
int client_id[MAX_CLI];
int count;

tpsvrinit(int argc, char *argv[])
{
    num_cli = 0;
```

```

    count = 0;
    printf("UCS Type Server tpsvrinit() is call\n");
}

/* TmaxのUCSモードでmainと同じ部分 */
int usermain(int argc, char *argv[])
{
    int    jobs;
    int    i;
    int    ret;
    char    *sndbuf;
    static int count = 0;

    printf("usermain start\n");

    sndbuf = (char *)tpalloc("CARRAY", NULL, 1024);

    while(1)
    {
        sleep(5);
        printf ("loop execute... %d\n", count);

        for (i = 0; i < num_cli; i++)
        {
            sprintf(sndbuf, "Success tpsendtocli [%d]", count++);

            /* クライアントIDを参照し、顧客にデータを送信する部分 */
            tpsendtocli (client_id[i], sndbuf, 1024, 0);
        }
        jobs = tpschedule(-1); /* while 最後に必ず必要 */
    }
}

LOGIN(TPSVCINFO *msg)
{
    char    *sndbuf;
    int     clid;
    int     ret;
    int     i;

    sndbuf = (char *)tpalloc("CARRAY", NULL, 1024);

    if (num_cli < MAX_CLI)
    {
        /* クライアントのID値を保存する部分 */
        client_id[num_cli] = tpgetclid();
        printf("client id(clid) = %d\n", client_id[num_cli]);
    }
}

```

```

        num_cli++;
    }
    sprintf(sndbuf, "Client Registration Success");

    tpreturn(TPSUCCESS, 0, (char *)sndbuf, 1000, 0);
}
tpsvrdone()
{
}

```

結果

以下は、サーバー・プログラムの例の結果です。

```

usermain start
UCS Type Server tpsvrinit() is call
usermain start
loop execute... 0
loop execute... 0
loop execute... 0
client id(clid) = 2097152
loop execute... 0
loop execute... 1
loop execute... 2
loop execute... 3
loop execute... 4
loop execute... 5
loop execute... 6
loop execute... 7
loop execute... 8
loop execute... 9
loop execute... 10
loop execute... 11
loop execute... 12
loop execute... 13
loop execute... 14
client id(clid) = 2097153
loop execute... 15
loop execute... 17
loop execute... 19
....

```


4.5. サーバー対サーバー・プログラム

以下は、3つのサーバー・プロセスを利用し、1つのプロセスはUCSでループを繰り返し、もう1つはデータベースからデータを選択して別のプロセスを呼び出して他の表に挿入するサンプル・プログラムです。

1. <ucssvr.c>はUCSプロセスで、mainsvr.pcにあるMAINというサービスに対してtpcallを繰り返すプログラムです。
2. UCSプロセスからtpcallを受信した<mainsvr.pc>は、データベースにあるtest_selという表から最初のデータを選択し、<inssvr.pc>にあるINSというサービスをtpcallします。
3. tpcallを受信した<inssvr.pc>は、test_insという表にtest_sel表から選択したデータを挿入します。このプロセスが行われると、<mainsvr.pc>はtx_commit()を行い、test_sel表からinssvr.pcに渡したデータを削除します。
4. test_sel表に6つのデータがあるため、同じプロセスを6回繰り返します。
5. 削除するデータがないため、1403エラーが発生します。

設定ファイル

```
*DOMAIN
tmax1          SHMKEY =79970, MINCLH=1, MAXCLH=3,
TPORTNO=8844, BLOCKTIME=120

*NODE
tmaxs2          TMAXDIR = "/user/jaya/tmax3511",
                APPDIR  = "/user/jaya/tmax3511/appbin",
                PATHDIR = "/user/jaya/tmax3511/path",
                TLOGDIR  = "/user/jaya/tmax3511/log/tlog",
                ULOGDIR  = "/user/jaya/tmax3511/log/ulog",
                SLOGDIR  = "/user/jaya/tmax3511/log/slog"

*SVRGROUP
tmaxs2_nx       NODENAME = "tmaxs2"
### tms for Oracle ###
tmaxs2_xa       NODENAME = "tmaxs2", DBNAME = ORACLE,
                OPENINFO = "Oracle_XA+Acc=P/scott/tiger+SesTm=60",
                TMSNAME  = tms_ora

*SERVER
ucssvr          SVGNAME = tmaxs2_nx, SVRTYPE = UCS, MIN = 1
mainsvr         SVGNAME = tmaxs2_xa, MIN = 1
inssvr          SVGNAME = tmaxs2_xa, MIN = 1
```

```

*SERVICE
MAIN          SVRNAME = mainsvr
INS           SVRNAME = ins

```

test_sel表

```

SQL> select * from test_sel;
A
-----
aaaaa
bbbbbb
cccccc
dddddd
eeeeee
ffffff

6 rows selected.

SQL> select * from test_ins;

no rows selected

```

サーバー・プログラム

<UCSSVR.C>

```

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

int usermain(int argc, char *argv[]) /* TmaxのUCSモードでmainと同じ部分 */
{
    int      ret;
    int      jobs;
    long     len;
    char     *sndbuf, *rcvbuf;

    printf("usermain start\n");
    sndbuf = (char *)tpalloc("STRING", NULL, 0);
    rcvbuf = (char *)tpalloc("STRING", NULL, 0);
    while(1) {

```

```

        ret = tpcall("MAIN", sndbuf, 0, &rcvbuf, &len, 0);
        if (ret == -1) {
            error processing...
        }
        jobs = tpschedule(-1);
        sleep (10);
    }
}

```

<mainsvr.pc>

```

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

EXEC SQL include sqlca.h;

#define MAXROW 6
EXEC SQL begin declare section;
    varchar v_a[MAXROW][11];
EXEC SQL end declare section;

MAIN(TPSVCINFO *msg)
{
    char *sndbuf, *rcvbuf;
    int i=0, errno;
    long len, ret;

    printf("[mainsvr] START\n");

    printf("[mainsvr] CURSOR DECLARE\n");
    EXEC SQL DECLARE cur_test_sel CURSOR FOR
    SELECT NVL(a, ' ')
    FROM test_sel
    WHERE rownum <= 6;

    printf("[mainsvr] CURSOR OPEN\n");
    EXEC SQL OPEN cur_test_sel;

    printf("[mainsvr] open cursor error : %d\n", sqlca.sqlcode);
    if ( sqlca.sqlcode != 0 ){
        error processing...
    }

    printf("[mainsvr] CURSOR FETCH\n");
    EXEC SQL FETCH cur_test_sel into :v_a;

```

```

if (sqlca.sqlcode < 0) {
    errno = sqlca.sqlcode;
    printf("[mainsvr] Fetch error : %d", errno);
    printf("[mainsvr] CURSOR CLOSE\n");
    EXEC SQL CLOSE cur_test_sel;
    printf("[mainsvr] TPRETURN FAIL\n");
    tpreturn( TPFAIL, errno, (char *)NULL, 0, 0 );
}
printf("[mainsvr] CURSOR CLOSE\n");
EXEC SQL CLOSE cur_test_sel;

for(i=0; i<MAXROW; i++) {
    sndbuf = (char *)tpalloc("STRING", 0, 0);
    rcvbuf = (char *)tpalloc("STRING", 0, 0);

    v_a[i].arr[v_a[i].len] = 0;
    strcpy(sndbuf, v_a[i].arr);
    printf("[mainsvr] %d : %s / %s\n", i, v_a[i].arr, sndbuf);

    printf("[mainsvr] TX_BEGIN\n");
    ret = tx_begin();
    if (ret < 0) {
        error processing...
    }
    printf("[mainsvr] INSERT\n");
    if(tpcall("INS", sndbuf, strlen(sndbuf), &rcvbuf, &len, 0)<0) {
        error processing...
    }
    else { /* Success */
        printf("[mainsvr] TX_COMMIT\n");
        EXEC SQL DELETE FROM TEST_SEL
            WHERE A = :sndbuf;
        if(sqlca.sqlcode != 0) {
            printf("[mainsvr] delete error : %d\n",
                sqlca.sqlcode);
            ret = tx_rollback();
            if (ret < 0) {
                error processing...
            }
            error processing...
        }
        tpfree(sndbuf);
        tpfree(rcvbuf);
        ret = tx_commit();
        if (ret < 0) {
            error processing...
        }
    }
}

```

```

        }
    }
    ...
    tpreturn( TPSUCCESS, 0, (char *)NULL, 0, 0 );
}

```

<inssvr.pc>

```

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

EXEC SQL include sqlca.h;

INS( TPSVCINFO *msg )
{
    char *buf;
    int i=0, errno;
    long len;

    printf("woong : START\n");
    buf = (char *)msg->data;

    printf("%s\n", buf);
    printf("woong : INSERT\n");
    EXEC SQL INSERT INTO TEST_INS VALUES(:buf);
    if (sqlca.sqlcode != 0 ) {
        error processing...
    }
    ...
    fflush(stdout);
    tpreturn(TPSUCCESS, 0, (char *)NULL, 0, 0 );
}

```

結果

サーバー・プログラムの例の結果は以下のとおりです。

```

[mainsvr] START
[mainsvr] CURSOR DECLARE
[mainsvr] CURSOR OPEN
[mainsvr] open cursor error : 0
[mainsvr] CURSOR FETCH
[mainsvr] CURSOR CLOSE
[mainsvr] 0 : aaaaa / aaaaa

```

```

[mainsvr] TX_BEGIN
[mainsvr] INSERT
woong : START
aaaaa
woong : INSERT
woong : TPRETURN SUCCESS
[mainsvr] TX_COMMIT
[mainsvr] 1 : bbbbb / bbbbb
[mainsvr] TX_BEGIN
[mainsvr] INSERT
woong : START
bbbbbb
woong : INSERT
woong : TPRETURN SUCCESS
[mainsvr] TX_COMMIT
[mainsvr] 2 : ccccc / ccccc
[mainsvr] TX_BEGIN
[mainsvr] INSERT
woong : START
ccccc
woong : INSERT
woong : TPRETURN SUCCESS
[mainsvr] TX_COMMIT
[mainsvr] 3 : ddddd / ddddd
[mainsvr] TX_BEGIN
[mainsvr] INSERT
woong : START
dddddd
woong : INSERT
woong : TPRETURN SUCCESS
[mainsvr] TX_COMMIT
[mainsvr] 4 : eeeee / eeeee
[mainsvr] TX_BEGIN
[mainsvr] INSERT
woong : START
eeeeee
woong : INSERT
woong : TPRETURN SUCCESS
[mainsvr] TX_COMMIT
[mainsvr] 5 : fffff / fffff
[mainsvr] TX_BEGIN
[mainsvr] INSERT
woong : START
ffffff
woong : INSERT
woong : TPRETURN SUCCESS
[mainsvr] TX_COMMIT.....6番目が終了し、それ以上選択されるデータがない

```

```

[mainsvr] TPRETURN SUCCESS
[mainsvr] START
[mainsvr] CURSOR DECLARE
[mainsvr] CURSOR OPEN
[mainsvr] open cursor error : 0
[mainsvr] CURSOR FETCH
[mainsvr] CURSOR CLOSE
[mainsvr] 0 : aaaaa / aaaaa
[mainsvr] TX_BEGIN
[mainsvr] INSERT
woong : START
aaaaa
woong : INSERT
woong : TPRETURN SUCCESS
[mainsvr] TX_COMMIT
[mainsvr] delete error : 1403
[mainsvr] TPRETURN FAIL
[mainsvr] START
[mainsvr] CURSOR DECLARE
[mainsvr] CURSOR OPEN
[mainsvr] open cursor error : 0
[mainsvr] CURSOR FETCH
[mainsvr] CURSOR CLOSE
[mainsvr] 0 : aaaaa / aaaaa
[mainsvr] TX_BEGIN
[mainsvr] INSERT
woong : START
aaaaa
woong : INSERT
woong : TPRETURN SUCCESS
[mainsvr] TX_COMMIT
[mainsvr] delete error : 1403
[mainsvr] TPRETURN FAIL
.....
.....
.....

```

test_ins表

各表を選択すると、以下のような結果が表示されます。

```

SQL> select * from test_ins;

NAME
-----
bbbbbb

```

```
cccc
```

```
dddd
```

```
eeee
```

```
ffff
```

```
aaaa
```

```
6 rows selected.
```

```
SQL> select * from test_sel;
```

```
no rows selected
```


第5章 RDPプログラム

本章では、RDPプログラムの構成、環境設定、コンパイル方法について説明します。

5.1. 概要

RDPはUCSと同じ形式のサーバーで動作します。RDPは、持続的に変化するデータを効率的かつスピーディーにクライアントに伝達するために、UCSタイプのプロセスをカーネルレベルで改良したプロセスです。

RDPはCLHを経由せずにクライアントにデータを伝達するため、少量のデータを多数のクライアントに短時間の間隔で送信した場合に、プロセス占有率や処理速度においてUCSより優れた性能を発揮します。

5.2. RDPサーバー・プログラムの構成

RDPサーバー・プログラムは基本的にUCSプログラムのようにusermain()を通じてアプリケーション・ロジックを実装します。RDPサーバー・プログラムは以下のようなモジュールとライブラリーで構成されます。

- \$(TMAXDIR)/lib/libsvrrs. so

プログラムのmain()と、各種RDP関連APIなどを持っているライブラリーです。RDPプログラムをコンパイルする際、常にリンクする必要があります。

- int tpsvrinit(int argc, char *argv[])

プログラムを起動する際に1回実行されます。グローバル変数の初期化や非XAの場合にデータベースの接続などを実装します。

- int tpsvrdone()

プログラムを終了する際に1回実行されます。使用リソースのリターンや非XAの場合にデータベースの接続解除などを実装します。

- int usermain(int argc, char *argv[])

実際にアプリケーションのロジックが実装される部分です。大部分が無限ループ形式で実装されます(usermain()モジュールで返されると、tpsvrdone()を実行し、プロセスが終了する形式であるためです)。

UCSサーバー・プログラムとは異なり、RDPサーバー・プログラムではtpschedule()が必要ありません。RDPクライアント・プログラムはUCSクライアント・プログラムと同じです。tpsetunsol_flag()、tpsetunsol()、tpgetunsol()などのAPIを使用して非要求メッセージを受信できるようにプログラムを実装します。詳細については、「[2.2.2. クライアント・プログラム](#)」を参照してください。

5.3. RDP環境設定およびコンパイル

5.3.1. 環境設定

RDPのための環境ファイルを作成するには、まず、DOMAINセクションのMINCLHの数とMAXCLHの数を一致させる必要があります。その後、NODEセクションのREALSVR項目に実際のサーバー名とrscpc項目を設定します。

実サーバーは1つのノードに一意です。ノード内の他のサーバー・プロセスも実サーバーにデータを送信後、実サーバーからデータをクライアントに送信します。各サーバー・プロセスからのサービス結果値は実サーバーに送信されますが、使用されるチャンネル数はrscpcで設定する必要があります。

SERVERセクションのMINとMAXの数は、DOMAINセクションのMINCLHとMAXCLHの数の2倍程度にするのが一般的です。また、SVRTYPEは「REALSVR」と設定します。

```
*DOMAIN
tmax1          SHMKEY =70990, MINCLH=1, MAXCLH=1

*NODE
tmaxi1         TMAXDIR = "/home/navis/tmax",
                APPDIR  = "/home/navis/tmax/appbin",
                PATHDIR = "/home/navis/tmax/path",
                TLOGDIR = "/home/navis/tmax/log/tlog",
                ULOGDIR = "/home/navis/tmax/log/ulog",
                SLOGDIR = "/home/navis/tmax/log/slog",
                REALSVR = "real", rscpc = 16

*SVRGROUP
svg1           NODENAME = "tmaxi1"
svg2           NODENAME = "tmaxi1"

*SERVER
deal          SVGNAME = svg2, MIN=1
real          SVGNAME = svg1, MIN=2, MAX=2, SVRTYPE = REALSVR

*SERVICE
IN            SVRNAME = deal
OUT           SVRNAME = deal
```

5.3.2. コンパイル

RDPサーバー・プログラムは、コンパイルする際にRDPライブラリー(libsvrrs.so)とリンクしている必要があります。プログラム内にも\$TMAXDIR/usrinc/ucs.hが含まれる必要があります、MakefileのTMAXLIBSに必ず-lsvrrsと-lpthreadを含む必要があります。

以下は、32bit SolarisでRDPサーバー・プログラムUCSをコンパイルするためのMakefileの例です。

```
# Server makefile
TARGET = $(COMP_TARGET)
APOBJS = $(TARGET).o
SDLFILE = demo.s
#Solarisの場合
LIBS = -lsvrrs -lpthread -lnodb -lsocket -lnsl
OBS = $(APOBJS) $(SDLOBJ) $(SVCTOBJ)
SDLOBJ = ${SDLFILE:.s=_sdl.o}
SDLC = ${SDLFILE:.s=_sdl.c}
SVCTOBJ = $(TARGET)_svctab.o
CFLAGS = -O -I$(TMAXDIR)
APPDIR = $(TMAXDIR)/appbin
SVCTDIR = $(TMAXDIR)/svct
LIBDIR = $(TMAXDIR)/lib
#
.SUFFIXES : .c
.c.o:
$(CC) $(CFLAGS) -c $<
#
# server compile
#

$(TARGET): $(OBS)
    $(CC) $(CFLAGS) -L$(LIBDIR) -o $(TARGET) $(OBS) $(LIBS)
    mv $(TARGET) $(APPDIR)/.
    rm -f $(OBS)
$(APOBJS): $(TARGET).c
    $(CC) $(CFLAGS) -c $(TARGET).c
$(SVCTOBJ):
    touch $(SVCTDIR)/$(TARGET)_svctab.c
    $(CC) $(CFLAGS) -c $(SVCTDIR)/$(TARGET)_svctab.c
$(SDLOBJ):
    $(TMAXDIR)/bin/sdlc -i ../sdl/$(SDLFILE)
    $(CC) $(CFLAGS) -c ../sdl/$(SDLC)
#
clean:
    -rm -f *.o core $(TARGET)
```

参考

OSに応じてMakefileの内容が異なることがあります。

第6章 RDPの使用例

本章では、RDPプログラムの使用例について説明します。

6.1. クライアント対サーバー・プログラム

以下は、簡単なRDPのサンプル・プログラムです。

クライアントでREALサービス呼び出すと、クライアントのIDを共有メモリーに保存します。RDPでは共有メモリーに保存されたすべてのクライアントにメッセージを送ります。その後クライアントがUNREALというサービス呼び出すと、該当クライアントのIDを削除し、該当するクライアントにメッセージを送らなくなります。

設定ファイル

```
*DOMAIN
tmax1          SHMKEY=73060, MINCLH=2, MAXCLH=2, TPORTNO=8800

*NODE
tmaxs1         TMAXDIR = "/home/tmax ",
                APPDIR  = "/home/tmax/appbin",
                PATHDIR = "/home/tmax/path",
                TLOGDIR = "/home/tmax/log/tlog",
                ULOGDIR = "/home/tmax/log/ulog",
                SLOGDIR = "/home/tmax/log/slog",
                REALSVR="realtest",
                RSCPC   = 16

*SVRGROUP
svg1           NODENAME = "tmaxs1"

*SERVER
realsvr        SVGNAME = svg1
realtest       SVGNAME = svg1,
                MIN    = 2,
                MAX    = 2,
                SVRTYPE = REALSVR,
                MAXRSTART = 0

*SERVICE
```

```
# UCSで提供するTCS TypeのService
REAL                SVRNAME = ucssvr
UNREAL              SVRNAME = ucssvr
```

クライアント・プログラム

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

main(int argc, char *argv[])
{
    char *sndbuf;
    char *rcvbuf;
    long rcvlen;
    int RecvCnt = 0, ret;

    if(tpstart((TPSTART_T *)NULL) == -1)
    {
        error processing...
    }

    tpsetunsol_flag(TPUNSOL_POLL);

    if((sndbuf = (char *)tpalloc("CARRAY", NULL, 1024)) == NULL)
    {
        error processing...
    }

    if((rcvbuf = (char *)tpalloc("CARRAY", NULL, 1024)) == NULL)
    {
        error processing...
    }

    if(tpcall("REAL", sndbuf, 1024, &rcvbuf, &rcvlen, 0) == -1)
    {
        error processing...
    }

    while(1)
    {
        ret = tpgetunsol(UNSOL_TPSENDTOCLI, &rcvbuf, &rcvlen, TPBLOCK);
        printf("Loop Count : %d\n", RecvCnt);
        if(ret > 0)
```

```

        {
            printf("ret message..[%s]\n", rcvbuf);
            RecvCnt ++;
        }
        if (RecvCnt == 10) break;
    }

    if(tpcall("UNREAL", sndbuf, 1024, &rcvbuf, &rcvlen, 0) == -1)
    {
        error processing...
    }

    RecvCnt = 0;
    while(1)
    {
        ret = tpgetunsol(UNSOL_TPSENDTOCLI, &rcvbuf, &rcvlen, TPBLOCK);
        printf("Loop Count : %d\n", RecvCnt);
        if(ret > 0)
        {
            printf("ret message..[%s]\n", rcvbuf);
            RecvCnt ++;
        }
        if (RecvCnt == 10) break;
    }
    tpfree((char *)sndbuf);
    tpfree((char *)rcvbuf);
    tpend();
}

```

サーバー・プログラム

<realsvr.c>

```

#include      <stdio.h>
#include      <sys/types.h>
#include      <sys/ipc.h>
#include      <sys/shm.h>
#include      <sys/time.h>
#include      <usrinc/tmaxapi.h>
#include      <usrinc/ucs.h>

int shm_creator = 0;
int *shm;
int id;

```

```

#define TABLE_SIZE      (FD_SETSIZE * 10)

tpsvrinit(int argc, char *argv[])
{
    int i;

    id = shmget(0x12345, TABLE_SIZE * sizeof(int),
                IPC_CREAT | IPC_EXCL | SHM_R | SHM_W);
    if (id < 0) {
        id = shmget(0x12345, TABLE_SIZE * sizeof(int), 0);
        if (id < 0) {
            error processing...
        }
    } else {
        shm_creator = 1;
    }

    shm = (int *)shmat(id, 0, 0);
    if (shm == (void*)-1) {
        error processing...
    }

    if (shm_creator) {
        for (i = 0; i < TABLE_SIZE; i++) {
            shm[i] = -1;
        }
    }

    printf("svr20 started\n");
    return 1;
}

tpsvrdone()
{
    shmdt((char *)shm);
    if (shm_creator) {
        shmctl(id, IPC_RMID, (struct shmid_ds *) 0);
    }
    printf("svr20 closed\n");
}

REAL(TPSVCINFO *rqst)
{
    int i, clid;

    clid = tpgetclid();

```



```

        for (i = 0; i < TABLE_SIZE; i++) {
            if (shm[i] == clid) {
                shm[i] = -1;
            }
        }
        for (i = 0; i < TABLE_SIZE; i++) {
            if (shm[i] == -1) {
                shm[i] = clid;
                break;
            }
        }

        tpreturn(TPSUCCESS, 0, rqst->data, rqst->len, 0);
    }

UNREAL(TPSVCINFO *rqst)
{
    int i, clid;

    clid = tpgetclid();
    for (i = 0; i < TABLE_SIZE; i++) {
        if (shm[i] == clid) {
            shm[i] = -1;
        }
    }

    usleep(10000);

    tpreturn(TPSUCCESS, 0, rqst->data, rqst->len, 0);
}

```

<realtest.c>

```

#include    <stdio.h>
#include    <sys/types.h>
#include    <sys/ipc.h>
#include    <sys/shm.h>
#include    <sys/time.h>
#include    <sys/timeb.h>
#include    <usrinc/tmaxapi.h>
#include    <usrinc/ucs.h>

int shm_creator = 0;
int *shm;
int id;

```

```

#define TABLE_SIZE      (FD_SETSIZE * 10)
#define MSG_LEN          (1024 - 20)
#define SEND_INTERVAL    (100000)

long time_diff(struct timeval *a, struct timeval *b)
{
    long    sec, usec;

    sec = a->tv_sec - b->tv_sec;
    usec = a->tv_usec - b->tv_usec;

    return (sec * 1000000 + usec);
}

tpsvrinit(int argc, char *argv[])
{
    int i;

    id = shmget(0x12345, TABLE_SIZE * sizeof(int),
               IPC_CREAT | IPC_EXCL | SHM_R | SHM_W);
    if (id < 0) {
        id = shmget(0x12345, TABLE_SIZE * sizeof(int), 0);
        if (id < 0) {
            error processing...
        }
    } else {
        shm_creator = 1;
    }

    shm = (int *)shmat(id, 0, 0);
    if (shm == (void*)-1) {
        error processing...
    }

    if (shm_creator) {
        for (i = 0; i < TABLE_SIZE; i++) {
            shm[i] = -1;
        }
    }

    printf("RealSvr started\n");
    return 1;
}

tpsvrdone()
{
    shmdt((char *)shm);
}

```

```

        if (shm_creator) {
            shmctl(id, IPC_RMID, (struct shmid_ds *) 0);
        }
        printf("RealSvr closed\n");
    }
}

usermain(int argc, char *argv[])
{
    char    *sndbuf;
    long    sndlen;
    int      i, n, msgid, previd, clid;
    int      fail, delay, loop;
    long     diff;
    struct timeval cur, prev, prev_stat;
    int      max, mine;

    max = tpgetminsvr();
    mine = tpgetsvrseqno();

    if ((sndbuf = (char *)tpalloc("CARRAY", NULL, 2048)) == NULL) {
        error processing...
    }

    msgid = previd = fail = delay = 0;
    gettimeofday(&prev, NULL);
    prev_stat = prev;
    while(1) {
        gettimeofday(&cur, NULL);

        if (mine == (max - 1)) {
            diff = time_diff(&cur, &prev_stat);
            if (diff >= 3975000) {
                printf("RealSvr: Sent = %d, fail = %d, delay = %d, "
                    "int = %d:%06d\n",
                    msgid - previd, fail, delay,
                    diff / 1000000, diff % 1000000);
                previd = msgid;
                delay = fail = 0;
                prev_stat = cur;
            }
        }

        diff = time_diff(&cur, &prev);
        if (diff < SEND_INTERVAL)
            tpuschedule(SEND_INTERVAL - diff);
        else
            delay++;
    }
}

```

```

        if (diff >= (SEND_INTERVAL * 2))
            printf("Long Schedule delay %d\n", diff);

        gettimeofday(&prev, NULL);

        for (i = 0; i < TABLE_SIZE; i++) {
            clid = shm[i];

            if (tpchkclid(clid) < 0) {
                continue;
            }

            sprintf(sndbuf, "Msg(%d) sent to clid = %#x", msgid++, clid);
            n = tpsendtocli(clid, sndbuf, MSG_LEN, TPFLOWCONTROL);
            if (n < 0) {
                fail++;
            }
        }
    }
}

```

索引

P

POD, 2

PODプロセス, 2

R

RDP, 4, 71

T

TCS, 1

TCSプロセス, 1

tpclrfd, 21

tpgetunsol, 40

tpissetfd, 23

tppost, 14

tpregcb, 25

tprelay, 27

tpsavectx, 29

tpschedule, 32

tpsendtocli, 33

tpsetfd, 15

tpsetunsol, 43

tpsetunsol_flag, 45

tpsubscribe, 18

tpsvctimeout, 35

tpsvrdone, 36

tpsvrdown, 38

tpunregcb, 39

tpunsubscribe, 20

U

UCS, 2, 5

