

IBM

@server

iSeries

Sun TI-RPC 分散アプリケーション

バージョン 5 リリース 3





@server

iSeries

Sun TI-RPC 分散アプリケーション

バージョン 5 リリース 3

ご注意

本書および本書で紹介する製品をご使用になる前に、39 ページの『特記事項』に記載されている情報をお読みください。

本書は、Operating System/400® (5722-SS1) のバージョン 5、リリース 3、モディフィケーション 0、および改訂版で明記されていない限り、以降のすべてのリリースおよびモディフィケーションに適用されます。このバージョンは、すべての RISC モデルで稼動するとは限りません。また CISC モデルでは稼働しません。

本マニュアルに関するご意見やご感想は、次の URL からお送りください。今後の参考にさせていただきます。

<http://www.ibm.com/jp/manuals/main/mail.html>

なお、日本 IBM 発行のマニュアルはインターネット経由でもご購入いただけます。詳しくは

<http://www.ibm.com/jp/manuals/> の「ご注文について」をご覧ください。

(URL は、変更になる場合があります)

お客様の環境によっては、資料中の円記号がバックスラッシュと表示されたり、バックスラッシュが円記号と表示されたりする場合があります。

原 典： @server
iSeries
Sun TI-RPC Distributed Applications
Version 5 Release 3

発 行： 日本アイ・ビー・エム株式会社

担 当： ナショナル・ランゲージ・サポート

第1刷 2005.8

この文書では、平成明朝体™W3、平成明朝体™W7、平成明朝体™W9、平成角ゴシック体™W3、平成角ゴシック体™W5、および平成角ゴシック体™W7を使用しています。この(書体*)は、(財)日本規格協会と使用契約を締結し使用しているものです。フォントとして無断複製することは禁止されています。

注* 平成明朝体™W3、平成明朝体™W7、平成明朝体™W9、平成角ゴシック体™W3、平成角ゴシック体™W5、平成角ゴシック体™W7

© Copyright International Business Machines Corporation 1998, 2005. All rights reserved.

© Copyright IBM Japan 2005

目次

Sun TI-RPC を使用して分散アプリケーションを開発する 1

トピックの印刷	1
rpcbind デーモンを使用する	2
OS/400 で rpcbind デーモンが実行しているかを確認する	2
OS/400 で rpcbind デーモンを開始および終了する	2
rpcgen コンパイラーを使用する	3
ネットワーク選択メカニズムを使用する	3
データ変換サポートを使用する	3
例: TI-RPC コードに基づいてサービス・アプリケーションを開発する	4
例: TI-RPC 簡易レベル・サービス API	4
例: TI-RPC トップレベル・サービス API	7
例: TI-RPC 中間レベル・サービス API	14
例: TI-RPC エキスパート・レベル・サービス API	15

例: TI-RPC サービスへの認証の追加	16
TI-RPC コード例に基づいてクライアント・アプリケーションを開発する	17
例: TI-RPC 簡易レベル・クライアント API	17
例: TI-RPC トップレベル・クライアント API	22
例: TI-RPC 中間レベル・クライアント API	26
例: TI-RPC エキスパート・レベル・クライアント API	31
例: TI-RPC クライアントへの認証の追加	36
TI-RPC に関するその他の情報	37

付録. 特記事項. 39


商標	40
資料に関するご使用条件	41
コードの特記事項情報	41

Sun TI-RPC を使用して分散アプリケーションを開発する

リモート・プロシージャ・コール (RPC) は高水準のパラダイムを提供しており、分散アプリケーションを相互に通信させることができます。Sun Microsystems 社が開発したオープン・ネットワーク・コンピューティング (ONC) RPC を使用すると、クライアント・アプリケーションをサーバー・メカニズムから容易に分離して分散することができます。トランスポート層に依存しないリモート・プロシージャ・コール (TI-RPC) または ONC+ RPC は、リリースされている RPC の最新バージョンです。TI-RPC では、ネットワーク層で使用される、基礎となるプロトコルを抽出する方法を提供することにより、プロトコル同士でより途切れの少ない伝送を行うことができます。

AS/400® 上で TI-RPC を使用して分散アプリケーションを開発するには、以下のトピックを参照してください。

- 2 ページの『rpcbind デーモンを使用する』
- 3 ページの『rpcgen コンパイラーを使用する』
- 3 ページの『ネットワーク選択メカニズムを使用する』
- 3 ページの『データ変換サポートを使用する』

TI-RPC を使用して分散アプリケーションを設計、インプリメント、および保守するための詳細は、Sun Microsystems 社の ONC+ Developer's Guide を、Sun Product Documentation Web サイト  で参照してください。

コーディングの例

サービスおよびクライアント・アプリケーション・プログラミング・インターフェースの詳細については、以下のトピックを参照してください。

- 4 ページの『例: TI-RPC コードに基づいてサービス・アプリケーションを開発する』
- 17 ページの『TI-RPC コード例に基づいてクライアント・アプリケーションを開発する』

トピックの印刷

この文書の PDF 版を参照用または印刷用にダウンロードし、表示することができます。PDF ファイルを表示したり印刷したりするには、Adobe® Acrobat® Reader が必要です。これは、Adobe Web サイト

(<http://www.adobe.com/prodindex/acrobat/readstep.html>)  からダウンロードできます。

PDF 版をダウンロードし、表示するには、『Sun TI-RPC 分散アプリケーション』(約 334 KB) を選択します。

表示用または印刷用の PDF をワークステーションに保存するには、次のようにします。

1. ブラウザーで PDF を開く (上記のリンクをクリックする)。
2. ブラウザーのメニューから「ファイル」をクリックする。
3. 「名前を付けて保存」をクリックする。
4. PDF を保管したいディレクトリーに進む。
5. 「保存」をクリックする。

rpcbind デーモンを使用する

クライアントがリモート・プロシージャ・コール (RPC) サービスへの接続を望む場合、RPC は RPCBIND デーモンと連絡を取り、サービスのアドレスを要求します。このようにして、アドレッシングが動的にできるため、クライアントはサービスが待機しているポートを知る必要がありません。サービスを利用するためには、rpcbind デーモンを使用して登録することが必要です。rpcbind デーモンが非活動状態である場合、サービスは起動できず、クライアントもサービスを見つけることができません。

OS/400® で、rpcbind デーモンを使用するには、以下のタスクを完了します。

1. rpcbind デーモンが OS/400 で実行しているかを確認します。
2. まだ実行していない場合は、rpcbind デーモンを OS/400 で開始してください。

OS/400 で rpcbind デーモンが実行しているかを確認する

トランスポート層に依存しないリモート・プロシージャ・コール (TI-RPC) のアプリケーション・プログラミング・インターフェース (API) を使用するには、rpcbind デーモン・ジョブ (QNFSRPCD) が OS/400 で実行している必要があります。

rpcbind デーモンが実行しているかを確認するには、以下のタスクを完了します。

1. OS/400 コマンド行で、WRKACTJOB と入力します。
2. サブシステム QSYSWRK で、次のジョブがあるかどうかを検索します。

```
QNFSRPCD The rpcbind daemon
```

rpcbind デーモンが実行されていない場合にそれを開始する際の説明については、『OS/400 で rpcbind デーモンを開始および終了する』を参照してください。

OS/400 で rpcbind デーモンを開始および終了する

rpcbind デーモン (RPCBIND) コマンドは rpcbind デーモン・ジョブ (QNFSRPCD) を開始します。

rpcbind デーモン・ジョブを開始する

rpcbind デーモン・ジョブを開始するには、次のコマンドを入力します。

```
RPCBIND RTVRPCREG(*YES)
```

注: このコマンドのオプション・パラメーターは RTVRPCREG で、rpcbind デーモンの開始時に、前に記録された登録情報を検索するかどうかを指定します。このパラメーターのデフォルトは *NO です。

rpcbind デーモンが開始時に登録情報を検索するようにしたい場合は *YES を選択します。このコマンドのパラメーターおよび値の説明についての詳細は、オンライン・ヘルプ・テキストを参照してください。

rpcbind デーモン・ジョブを終了する

rpcbind デーモン・ジョブ (QNFSRPCD) を終了するには、次のコマンドを入力します。

```
ENDRPCBIND
```

rpcgen コンパイラーを使用する

RPCGEN コマンドは、リモート・プロシージャ・コール言語 (RPCL) で書かれた入力ファイルから C コードを生成します。生成された C コードを使用して、RPC プロトコルをインプリメントします。

OS/400 で、rpcgen コンパイラーを使用するには、以下のタスクを完了します。

1. RPCL にソース入力ファイルを作成します。

rpcgen コンパイラーの使用法について詳しくは、Sun Product Documentation Web サイト  を参照してください。

2. 次のコマンドを入力して、rpcgen コンパイラーを OS/400 で実行します。

```
RPCGEN
```

注: このコマンドのパラメーターおよび値の説明については、オンライン・ヘルプ・テキストを参照してください。

3. OS/400 で C 言語 コンパイラーを使用し、rpcgen コンパイラーからの出力をコンパイルします。

注: AS/400 で、統合言語環境® (ILE) C コンパイラーを使用している場合は、出力ファイルをソース・メンバーとして保管する必要があります。

ネットワーク選択メカニズムを使用する

ネットワーク選択メカニズムにより、アプリケーションが実行する際のトランスポートを選択することができます。/etc/netconfig ファイルは、ホストに対して使用可能なトランスポートをリストし、タイプによって識別するデータベースです。トランスポートは、/etc/netconfig ファイルに示された指定の順序で使用できます。ネットワーク選択アプリケーション・プログラミング・インターフェース (API) についての詳細は、System API Reference を参照してください。

iSeries ナビゲーターで OS/400 の /etc/netconfig ファイルにアクセスしたい場合は、以下のステップを完了してください。

1. iSeries ナビゲーターがパーソナル・コンピューターにインストールされている場合は、それを開きます。
2. 「ネットワーク」フォルダーを展開します。
3. 「サーバー」フォルダーを展開します。
4. 「TCP/IP」をクリックします。
5. 「RPC」を右クリックし、ポップアップ・メニューから「プロパティ」を選択します。
6. 「トランスポート」タブをクリックします。

注: この情報を見るには、*IOSYSCFG 権限が必要です。

データ変換サポートを使用する

OS/400 の各国語サポート (NLS) では、トランスポート層に依存しないリモート・プロシージャ・コール (TI-RPC) のアプリケーション・プログラミング・インターフェース (API) がすべて使用可能になっています。このサポートは、外部データ表示 (XDR) 関数のリストに追加されています。これらの関数を使用して、異なるコード・ページにあるクライアントとサービスの間でデータの通信を行うことができます。システム管理者は、コード・ページをリモート・クライアントと関連付けるために、/etc/rpcnls ファイルを保持します。XDR 関数は、/etc/rpcnls ファイルの情報を使って暗黙のデータ変換を行います。次の XDR 関数は、暗黙のデータ変換ルーチンを組み込みます。

- xdr_char() (1 バイトのみ)
- xdr_u_char() (1 バイトのみ)
- xdr_double_char (1 バイトおよび 2 バイト)
- xdr_string() (1 バイトおよび 2 バイト)
- xdr_wrapstring () (1 バイトおよび 2 バイト)

各トランスポート層に依存しないリモート・プロシージャ・コール (TI-RPC) アプリケーション・プログラミング・インターフェース (API) のデータ変換サポートについての詳細は、System API Reference を参照してください。

iSeries ナビゲーターで OS/400 の /etc/rpcnls ファイルにアクセスしたい場合は、以下のステップを完了してください。

1. iSeries ナビゲーターがパーソナル・コンピューターにインストールされている場合は、それを開きます。
2. 「ネットワーク」フォルダーを展開します。
3. 「サーバー」フォルダーを展開します。
4. 「TCP/IP」をクリックします。
5. 「RPC」を右クリックし、ポップアップ・メニューから「プロパティ」を選択します。
6. 「データ変換」タブをクリックします。

注: この情報を見るには、*IOSYSCFG 権限が必要です。

例: TI-RPC コードに基づいてサービス・アプリケーションを開発する

トランスポート層に依存しないリモート・プロシージャ・コール (TI-RPC) プログラミングは、OS/400 上でクライアント/サーバー間の分散アプリケーションを開発するための効率的な方法を提供します。

OS/400 上でサービス・アプリケーションを開発するには、以下のコード例をガイドラインとして使用してください。

- 『例: TI-RPC 簡易レベル・サービス API』
- 7 ページの 『例: TI-RPC トップレベル・サービス API』
- 14 ページの 『例: TI-RPC 中間レベル・サービス API』
- 15 ページの 『例: TI-RPC エキスパート・レベル・サービス API』
- 16 ページの 『例: TI-RPC サービスへの認証の追加』

関連情報

- 1 ページの 『Sun TI-RPC を使用して分散アプリケーションを開発する』
- 17 ページの 『TI-RPC コード例に基づいてクライアント・アプリケーションを開発する』

例: TI-RPC 簡易レベル・サービス API

次のコード例では、トランスポート層に依存しないリモート・プロシージャ・コール (TI-RPC) サービスの開発に使用される、簡易レベル・サービス・アプリケーション・プログラミング・インターフェース (API) の 1 つを例示します。

このコード例では、各プロシージャが、他のプロシージャから独立して登録される方法に注目してください。簡易レベルでは、サービスに複数のプロシージャがある場合でも、それぞれを個別に登録しなければなりません。サービスが登録抹消される場合は、すべてのプロシージャが一度に抹消されます。他のプロシージャをそのままにしておいて、個々のプロシージャを登録抹消するという方法はありません。

このレベルは、プロシージャラーの数が少ないサービスに適しています。また、限られた数の最終プロシージャラーを使って、かなり大きなサービスのプロトタイプを作成するのも役立ちます。どのサービス・レベルでも、サービスの最終コールは、`svc_run()` でなければなりません。このコールで、`Select Wait` (クライアントからの接続を待機する) 状態になります。

```
#include <stdio.h>
#include <netconfig.h>
#include <rpc/rpc.h>
#include <errno.h>
#include "myapp.h"

int main(int argc, char *argv[]) {

    bool_t rslt; /* return value for rpc_call() */

    /* unregister any existing copy of this service */
    /* (void)svc_unreg(program, version) */
    svc_unreg(PROGNUM, VERSNUM);

    /* (bool_t)rpc_reg(prognum, versnum, procnum, procname, */
    /*                xdr_in, xdr_out, nettype) */
    rslt = rpc_reg(PROGNUM, VERSNUM, GET_UID, myapp_get_uid,
        xdr_wrapstring, xdr_u_int, NETTYPE);

    /* check for errors calling rpc_reg() */
    if (rslt == FALSE) {
        /* print error messages and exit */
        fprintf(stderr, "Error calling rpc_reg for %s\n", "GET_UID");
        fprintf(stderr, "PROG: %lu\tVERS: %lu\tNET: %s\n",
            PROGNUM, VERSNUM, NETTYPE);
        /* clean up before exiting */
        svc_unreg(PROGNUM, VERSNUM);
        return 1;
    }

    /* (bool_t)rpc_reg(prognum, versnum, procnum, procname, */
    /*                xdr_in, xdr_out, nettype) */
    rslt = rpc_reg(PROGNUM, VERSNUM, GET_UID_STRING, myapp_get_uid_string,
        xdr_wrapstring, xdr_wrapstring, NETTYPE);

    /* check for errors calling rpc_reg() */
    if (rslt == FALSE) {
        /* print error messages and exit */
        fprintf(stderr, "Error calling rpc_reg for %s\n", "GET_UID_STRING");
        fprintf(stderr, "PROG: %lu\tVERS: %lu\tNET: %s\n",
            PROGNUM, VERSNUM, NETTYPE);
        /* clean up before exiting */
        svc_unreg(PROGNUM, VERSNUM);
        return 1;
    }

    /* (bool_t)rpc_reg(prognum, versnum, procnum, procname, */
    /*                xdr_in, xdr_out, nettype) */
    rslt = rpc_reg(PROGNUM, VERSNUM, GET_SIZE, myapp_get_size,
        xdr_wrapstring, xdr_int, NETTYPE);

    /* check for errors calling rpc_reg() */
    if (rslt == FALSE) {
        /* print error messages and exit */
        fprintf(stderr, "Error calling rpc_reg for %s\n", "GET_SIZE");
        fprintf(stderr, "PROG: %lu\tVERS: %lu\tNET: %s\n",
            PROGNUM, VERSNUM, NETTYPE);
        /* clean up before exiting */
        svc_unreg(PROGNUM, VERSNUM);
        return 1;
    }
}
```

```

/* (bool_t)rpc_reg(prognum, versnum, procnum, procname, */
/*           xdr_in, xdr_out, nettype)           */
rslt = rpc_reg(PROGNUM, VERSNUM, GET_MTIME, myapp_get_mtime,
  xdr_wrapstring, xdr_long, NETTYPE);

/* check for errors calling rpc_reg() */
if (rslt == FALSE) {
  /* print error messages and exit */
  fprintf(stderr, "Error calling rpc_reg for %s\n", "GET_MTIME");
  fprintf(stderr, "PROG: %lu\tVERS: %lu\tNET: %s\n",
    PROGNUM, VERSNUM, NETTYPE);
  /* clean up before exiting */
  svc_unreg(PROGNUM, VERSNUM);
  return 1;
}

/* (bool_t)rpc_reg(prognum, versnum, procnum, procname, */
/*           xdr_in, xdr_out, nettype)           */
rslt = rpc_reg(PROGNUM, VERSNUM, GET_MTIME_STRING, myapp_get_mtime_string,
  xdr_wrapstring, xdr_wrapstring, NETTYPE);

/* check for errors calling rpc_reg() */
if (rslt == FALSE) {
  /* print error messages and exit */
  fprintf(stderr, "Error calling rpc_reg for %s\n", "GET_MTIME_STRING");
  fprintf(stderr, "PROG: %lu\tVERS: %lu\tNET: %s\n",
    PROGNUM, VERSNUM, NETTYPE);
  /* clean up before exiting */
  svc_unreg(PROGNUM, VERSNUM);
  return 1;
}

/* (bool_t)rpc_reg(prognum, versnum, procnum, procname, */
/*           xdr_in, xdr_out, nettype)           */
rslt = rpc_reg(PROGNUM, VERSNUM, GET_CODEPAGE, myapp_get_codepage,
  xdr_wrapstring, xdr_u_short, NETTYPE);

/* check for errors calling rpc_reg() */
if (rslt == FALSE) {
  /* print error messages and exit */
  fprintf(stderr, "Error calling rpc_reg for %s\n", "GET_CODEPAGE");
  fprintf(stderr, "PROG: %lu\tVERS: %lu\tNET: %s\n",
    PROGNUM, VERSNUM, NETTYPE);
  /* clean up before exiting */
  svc_unreg(PROGNUM, VERSNUM);
  return 1;
}

/* (bool_t)rpc_reg(prognum, versnum, procnum, procname, */
/*           xdr_in, xdr_out, nettype)           */
rslt = rpc_reg(PROGNUM, VERSNUM, GET_OBJTYPE, myapp_get_objtype,
  xdr_wrapstring, xdr_wrapstring, NETTYPE);

/* check for errors calling rpc_reg() */
if (rslt == FALSE) {
  /* print error messages and exit */
  fprintf(stderr, "Error calling rpc_reg for %s\n", "GET_OBJTYPE");
  fprintf(stderr, "PROG: %lu\tVERS: %lu\tNET: %s\n",
    PROGNUM, VERSNUM, NETTYPE);
  /* clean up before exiting */
  svc_unreg(PROGNUM, VERSNUM);
  return 1;
}

/* (bool_t)rpc_reg(prognum, versnum, procnum, procname, */
/*           xdr_in, xdr_out, nettype)           */

```

```

rslt = rpc_reg(PROGNUM, VERSNUM, GET_FILETYPE, myapp_get_filetype,
    xdr_wrapstring, xdr_wrapstring, NETTYPE);

/* check for errors calling rpc_reg() */
if (rslt == FALSE) {
    /* print error messages and exit */
    fprintf(stderr, "Error calling rpc_reg for %s\n", "GET_FILETYPE");
    fprintf(stderr, "PROG: %lu\tVERS: %lu\tNET: %s\n",
        PROGNUM, VERSNUM, NETTYPE);
    /* clean up before exiting */
    svc_unreg(PROGNUM, VERSNUM);
    return 1;
}

/* (bool_t)rpc_reg(prognum, versnum, procnum, procname, */
/*                xdr_in, xdr_out, nettype) */
rslt = rpc_reg(PROGNUM, VERSNUM, END_SERVER, myapp_end_server,
    (xdrproc_t)xdr_void, (xdrproc_t)xdr_void, NETTYPE);

/* check for errors calling rpc_reg() */
if (rslt == FALSE) {
    /* print error messages and exit */
    fprintf(stderr, "Error calling rpc_reg for %s\n", "END_SERVER");
    fprintf(stderr, "PROG: %lu\tVERS: %lu\tNET: %s\n",
        PROGNUM, VERSNUM, NETTYPE);
    /* clean up before exiting */
    svc_unreg(PROGNUM, VERSNUM);
    return 1;
}

/* this should loop indefinitely waiting for client connections */
svc_run();

/* if we get here, svc_run() returned */
fprintf(stderr, "svc_run() returned. ERROR has occurred.\n");
fprintf(stderr, "errno: %d\n", errno);

/* clean up by unregistering. then, exit */
svc_unreg(PROGNUM, VERSNUM);

return 1;
} /* end of main() */

```

例: TI-RPC トップレベル・サービス API

次のコード例では、トランスポート層に依存しないリモート・プロシージャ・コール (TI-RPC) サービスの開発に使用されるトップレベル・サービス・アプリケーション・プログラミング・インターフェース (API) を例示します。

トップレベルでサービスを開発するには、開発者がディスパッチ・ルーチンを作成する必要があるため、より複雑な作業になります。このレベルでは、サービス要求が入ってくるとディスパッチ・ルーチンが呼び出されます。ディスパッチ・ルーチンは引き数を収集して正しいローカル・プロシージャを呼び出し、すべてのエラーと結果を捕らえ、その情報をクライアントに戻す必要があります。ディスパッチ関数があったん作成されると、わずかな変更だけで他のサービスでコピーしたり使ったりすることが容易になります。

トップ、中間、およびエキスパート層は、変更せずに同じディスパッチ関数を使用することができます。次の例では、ディスパッチ関数は、このファイルの他のローカル関数と組み込まれています。両方のファイルを、サービスの実行前にコンパイルし、一緒にリンクすることが必要です。他の層の上にトップレベルをも

ってくることにより、ネットワーク選択 API を使用しないで、ネット・タイプをストリングとして指定できるという利点があります。トップレベル API を呼び出した後、rpcbind デーモンを使用してサービスを作成し、ディスパッチ関数に結合し登録します。

```
#include <stdio.h>
#include <netconfig.h>
#include <rpc/rpc.h>
#include <errno.h>
#include "myapp.h"
int main(int argc, char *argv[]) {

    int num_svc; /* return value for the svc_create() API */

    /* unregister any existing copy of this service */
    /* (void)svc_unreg(program, version) */
    svc_unreg(PROGNUM, VERSNUM);

    /* (int)svc_create(dispatch, prognum, versnum, nettype); */
    num_svc = svc_create(myapp_dispatch, PROGNUM, VERSNUM, NETTYPE);

    /* check for errors calling svc_create() */
    if (num_svc == 0) {
        /* print error messages and exit */
        fprintf(stderr, "Error calling %s.\n", "svc_create");
        fprintf(stderr, "PROG: %lu\nVERS: %lu\nNET: %s\n",
            PROGNUM, VERSNUM, NETTYPE);
        fprintf(stderr, "errno: %d\n", errno);
        return 1;
    }

    /* this should loop indefinitely waiting for client connections */
    svc_run();

    /* if we get here, svc_run() returned */
    fprintf(stderr, "svc_run() returned. ERROR has occurred.\n");
    fprintf(stderr, "errno: %d\n", errno);

    /* clean up by unregistering. then, exit */
    svc_unreg(PROGNUM, VERSNUM);

    return 1;
} /* end of main() */

/* This is an example of the dispatch function */
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <pwd.h>
#include <rpc/rpc.h>
#include <time.h>
#include "myapp.h"

char * myapp_get_uid(char *in) {

    u_int retval;          /* return value for this procedure() */
    struct stat sbuf;      /* data storage area for stat() */
    int stat_ret;          /* return value for stat() */
    char *file = *(char **)in; /* input value for stat() */

    /* (int)stat(filename, struct stat *) */
    stat_ret = stat(file, &sbuf);

    if (stat_ret == -1) {
        retval = (u_int)-1;
    }
}
```

```

else {
    retval = (u_int)(sbuf.st_uid);
}

return (char *)&retval;
}

char *myapp_get_uid_string(char *in) {

    char *retval;          /* return value for this procedure() */
    struct passwd *pbuf;   /* return value for getpwuid() */
    struct stat sbuf;     /* data storage area for stat() */
    int stat_ret;         /* return value for stat() */
    char *file = *(char **)in; /* input value for stat() */

    /* (int)stat(filename, struct stat *) */
    stat_ret = stat(file, &sbuf);

    if (stat_ret == -1) {
        retval = (char *)NULL;
    }
    else {

        pbuf = (struct passwd *)getpwuid((uid_t)(sbuf.st_uid));

        if (pbuf == NULL) {
            retval = (char *)NULL;
        }
        else {
            retval = (char *)(pbuf->pw_name);
        }
    }

    return (char *)&retval;
}

char * myapp_get_size(char *in) {

    int retval;          /* return value for this procedure() */
    struct stat sbuf;   /* data storage area for stat() */
    int stat_ret;      /* return value for stat() */
    char *file = *(char **)in; /* input value for stat() */

    /* (int)stat(filename, struct stat *) */
    stat_ret = stat(file, &sbuf);

    if (stat_ret == -1) {
        retval = (int)-1;
    }
    else {
        retval = (int)(sbuf.st_size);
    }

    return (char *)&retval;
}

char * myapp_get_mtime(char *in) {

    long retval;          /* return value for this procedure() */
    struct stat sbuf;   /* data storage area for stat() */
    int stat_ret;      /* return value for stat() */
    char *file = *(char **)in; /* input value for stat() */

    /* (int)stat(filename, struct stat *) */
    stat_ret = stat(file, &sbuf);

    if (stat_ret == -1) {

```

```

    retval = (long)-1;
}
else {
    retval = (long)(sbuf.st_mtime);
}

return (char *)&retval;
}

char *myapp_get_mtime_string(char *in) {

    char *retval;          /* return value for this procedure() */
    struct stat sbuf;      /* data storage area for stat() */
    int stat_ret;         /* return value for stat() */
    char *file = *(char **)in; /* input value for stat() */

    /* (int)stat(filename, struct stat *) */
    stat_ret = stat(file, &sbuf);

    if (stat_ret == -1) {
        retval = (char *)NULL;
    }

    else {
        retval = (char *)ctime((time_t *)&(sbuf.st_mtime));
    }

    return (char *)&retval;
}

char * myapp_get_codepage(char *in) {

    u_short retval;       /* return value for this procedure() */
    struct stat sbuf;     /* data storage area for stat() */
    int stat_ret;        /* return value for stat() */
    char *file = *(char **)in; /* input value for stat() */

    stat_ret = stat(file, &sbuf);

    if (stat_ret == -1) {
        retval = (u_short)-1;
    }

    else {
        retval = (u_short)(sbuf.st_codepage);
    }

    return (char *)&retval;
}

char *myapp_get_objtype(char *in) {

    char *retval;          /* return value for this procedure() */
    struct stat sbuf;      /* data storage area for stat() */
    int stat_ret;         /* return value for stat() */
    char *file = *(char **)in; /* input value for stat() */

    /* (int)stat(filename, struct stat *) */
    stat_ret = stat(file, &sbuf);

    if (stat_ret == -1) {
        retval = (char *)NULL;
    }

    else {

```



```

    retval = (char *)(sbuf.st_objtype);
}

return (char *)&retval;
}

char *myapp_get_filetype(char *in) {

char *result = NULL;      /* return value for this procedure() */
struct stat sbuf;        /* data storage area for stat() */
int stat_ret;           /* return value for stat() */
char *file = *(char **)in; /* input value for stat() */

/* (int)stat(filename, struct stat *) */
stat_ret = stat(file, &sbuf);

if (stat_ret == -1) {
    return (char *)NULL;
}

if (S_ISDIR(sbuf.st_mode)) {
    result = "Directory";
}

if (S_ISREG(sbuf.st_mode)) {
    result = "Regulare File";
}

if (S_ISLNK(sbuf.st_mode)) {
    result = "Symbolic Link";
}

if (S_ISSOCK(sbuf.st_mode)) {
    result = "Socket";
}

if (S_ISNATIVE(sbuf.st_mode)) {
    result = "AS/400 Native Object";
}

if (S_ISFIFO(sbuf.st_mode)) {
    result = "FIFO";
}

if (S_ISCHR(sbuf.st_mode)) {
    result = "Character Special";
}

if (S_ISBLK(sbuf.st_mode)) {
    result = "Block Special";
}

return (char *)&result;
}

char * myapp_end_server(char *empty) {

/* char *empty is not used */
/* function always returns NULL */

svc_unreg(PROGNUM, VERSNUM);

return (char *)NULL;
}

```

```

void myapp_dispatch(struct svc_req *request, SVCXPRT *svc) {

union {
/* all of the procedurs take a string */
/* if there were other procedures, it */
/* might look like this:          */
/* int set_codepage_arg           */
char * filename_arg;
} argument;

char *result;          /* pointer to returned data from proc */
xdrproc_t xdr_argument; /* decodes data from client call */
xdrproc_t xdr_result;  /* encodes data to return to client */
char *(*proc)(char *); /* pointer to local procedure to call */

switch (request->rq_proc) {

case NULLPROC:
/* a special case.  always return void */
(void)svc_sendreply((SVCXPRT *)svc,
(xdrproc_t)xdr_void,
(char *)NULL);

return;

case GET_UID:
/* takes a string argument (filename) */
/* returns an u_int (uid of file ownder) */
xdr_argument = xdr_wrapstring;
xdr_result   = xdr_u_int;
proc        = (char *(*)(char *))myapp_get_uid;
break;

case GET_UID_STRING:
/* takes a string argument (filename) */
/* returns a string (owner's name in string format) */
xdr_argument = xdr_wrapstring;
xdr_result   = xdr_wrapstring;
proc        = (char *(*)(char *))myapp_get_uid_string;
break;

case GET_SIZE:
/* takes a string argument (filename) */
/* returns an int (size of file in bytes) */
xdr_argument = xdr_wrapstring;
xdr_result   = xdr_int;
proc        = (char *(*)(char *))myapp_get_size;
break;

case GET_MTIME:
/* takes a string argument (filename) */
/* returns a long (time last modified) */
xdr_argument = xdr_wrapstring;
xdr_result   = xdr_long;
proc        = (char *(*)(char *))myapp_get_mtime;
break;

case GET_MTIME_STRING:
/* takes a string argument (filename) */
/* returns a string (time last modified, string format) */
xdr_argument = xdr_wrapstring;
xdr_result   = xdr_wrapstring;
proc        = (char *(*)(char *))myapp_get_mtime_string;
break;

case GET_CODEPAGE:
/* takes a string argument (filename) */

```

```

    /* returns an u_short (codepage of file) */
    xdr_argument = xdr_wrapstring;
    xdr_result   = xdr_u_short;
    proc        = (char *(*)(char *))myapp_get_codepage;
    break;

case GET_OBJTYPE:
    /* takes a string argument (filename) */
    /* returns a string (object type) */
    xdr_argument = xdr_wrapstring;
    xdr_result   = xdr_wrapstring;
    proc        = (char *(*)(char *))myapp_get_objtype;
    break;

case GET_FILETYPE:
    /* takes a string argument (filename) */
    /* returns a string (file type) */
    xdr_argument = xdr_wrapstring;
    xdr_result   = xdr_wrapstring;
    proc        = (char *(*)(char *))myapp_get_filetype;
    break;

case END_SERVER:
    /* takes no arguments */
    /* returns no data */
    /* unregisters service with local rpcbind daemon */
    xdr_argument = (xdrproc_t)xdr_void;
    xdr_result   = (xdrproc_t)xdr_void;
    proc        = (char *(*)(char *))myapp_end_server;
    break;

default:
    /* fall through case. return error to client */
    svcerr_noproc(svc);
    return;
} /* end switch(request->rq_proc) */

/* clear the argument */
memset((char *)&argument, (int)0, sizeof(argument));

/* decode argument from client using xdr_argument() */
if (svc_getargs(svc, xdr_argument, (char *)&argument) == FALSE) {
    /* if svc_getargs() fails, return RPC_CANTDECODEARGS to client */
    svcerr_decode(svc);
    return;
}

/* call local procedure, passing in pointer to argument */
result = (char *)(*proc)((char *)&argument);

/* check first that result isn't NULL */
/* try to send results back to client. check for failure */
if ((result != NULL) && (svc_sendreply(svc, xdr_result, result) == FALSE))
{
    /* send error message back to client */
    svcerr_systemerr(svc);
}

/* free the decoded argument's space */
if (svc_freeargs(svc, xdr_argument, (char *)&argument) == FALSE) {
    /* if unable to free, print error and exit */
    (void)fprintf(stderr, "unable to free arguments\n");
}

```

```

    exit(1);
}
} /* end of myapp_dispatch() */

```

例: TI-RPC 中間レベル・サービス API

次のコード例では、トランスポート層に依存しないリモート・プロシージャ・コール (TI-RPC) サービスの開発に使用される中間レベル・サービス・アプリケーション・プログラミング・インターフェース (API) を例示します。

ディスパッチ関数は、変更なしで再利用することができます。トップレベルと中間レベルの間の相違点は、ネットワーク選択 API を使用するかどうかということだけです。中間レベルでも `rpcbind` デーモンを使用してサービスを作成し、結合し登録します。

```

#include <stdio.h>
#include <netconfig.h>
#include <rpc/rpc.h>
#include <errno.h>
#include "myapp.h"

int main(int argc, char *argv[]) {

    struct netconfig *nconf; /* pointer to nettype data */
    SVCXPRT *svc;          /* pointer to service handle */

    /* unregister any existing copy of this service */
    /* (void)svc_unreg(program, version) */
    svc_unreg(PROGNUM, VERSNUM);

    /* (struct netconfig *)getnetconfigent(nettype) */
    nconf = getnetconfigent(NETTYPE);
    if (nconf == (struct netconfig *)NULL) {
        fprintf(stderr, "Error calling getnetconfigent(%s)%n", NETTYPE);
        fprintf(stderr, "errno: %d%n", errno);
        return 1;
    }

    /* (SVCXPRT *)svc_tp_create(dispatch, prognum, versnum, netconf) */
    svc = svc_tp_create(myapp_dispatch, PROGNUM, VERSNUM, nconf);

    /* check for errors calling svc_tp_create() */
    if (svc == (SVCXPRT *)NULL) {
        /* print error messages and exit */
        fprintf(stderr, "Error calling %s.%n", "svc_tp_create");
        fprintf(stderr, "PROG: %lu%tVERS: %lu%tNET: %s%t%n",
            PROGNUM, VERSNUM, NETTYPE);
        fprintf(stderr, "errno: %d%n", errno);
        return 1;
    }

    /* this should loop indefinitely waiting for client connections */
    svc_run();

    /* if we get here, svc_run() returned */
    fprintf(stderr, "svc_run() returned. ERROR has occurred.%n");
    fprintf(stderr, "errno: %d%n", errno);

    /* clean up by unregistering. then, exit */
    svc_unreg(PROGNUM, VERSNUM);

    return 1;
} /* end of main() */

```

例: TI-RPC エキスパート・レベル・サービス API

次のコード例では、トランスポート層に依存しないリモート・プロシージャ・コール (TI-RPC) サービスの開発に使用されるエキスパート・レベル・サービス・アプリケーション・プログラミング・インターフェース (API) を例示します。

エキスパート・レベルでは、プログラマーはサービスの送受信バッファ・サイズを指定することができます。 `svc_tli_create()` を呼び出すと、サービス・ハンドルが作成されますが、サービスを `rpcbind` デーモンを使用して登録したり結合したりすることはありません。プログラマーはサービスが適切に作動する前に、`svc_reg()` を呼び出す必要があります。中間レベルと同じように、ネットワーク選択 API を使用して、サービスに関するトランスポート情報を検索するオプションがあります。

```
#include <stdio.h>
#include <netconfig.h>
#include <rpc/rpc.h>
#include <errno.h>
#include "myapp.h"

int main(int argc, char *argv[]) {

    struct netconfig *nconf; /* pointer to nettype data */
    SVCXPRT *svc;          /* pointer to service handle */
    bool_t rslt;           /* return value for svc_reg() */

    /* unregister any existing copy of this service */
    /* (void)svc_unreg(program, version) */
    svc_unreg(PROGNUM, VERSNUM);

    /* (struct netconfig *)getnetconfigent(nettype) */
    nconf = getnetconfigent(NETTYPE);

    /* check for errors calling getnetconfigent() */
    if (nconf == (struct netconfig *)NULL) {
        /* print error messages and exit */
        fprintf(stderr, "Error calling getnetconfigent(%)%n", NETTYPE);
        fprintf(stderr, "errno: %d%n", errno);
        return 1;
    }

    /* (SVCXPRT *)svc_tli_create(filedes, netconfig, bindaddr, sendsz, recvsz) */
    svc = svc_tli_create(RPC_ANYFD, nconf, NULL, 0, 0);

    /* check for errors calling svc_tli_create() */
    if (svc == (SVCXPRT *)NULL) {
        /* print error messages and exit */
        fprintf(stderr, "Error calling %s.%n", "svc_tli_create");
        fprintf(stderr, "errno: %d%n", errno);
        return 1;
    }

    /* (bool_t)svc_reg(svcxprt, prognum, versnum, dispatch, netconf) */
    rslt = svc_reg(svc, PROGNUM, VERSNUM, myapp_dispatch, nconf);

    /* check for errors calling svc_reg() */
    if (rslt == FALSE) {
        /* print error messages and exit */
        fprintf(stderr, "Error calling svc_reg%n");
        fprintf(stderr, "PROG: %lu%nVERS: %lu%tNET: %s%n",
            PROGNUM, VERSNUM, NETTYPE);
        fprintf(stderr, "errno: %d%n", errno);
        return 1;
    }

    /* this should loop indefinitely waiting for client connections */
    svc_run();
}
```

```

/* if we get here, svc_run() returned */
fprintf(stderr, "svc_run() returned. ERROR has occurred.¥n");
fprintf(stderr, "errno: %d¥n", errno);

/* clean up by unregistering. then, exit */
svc_unreg(PROGNUM, VERSNUM);

return 1;

} /* end of main() */

```

例: TI-RPC サービスへの認証の追加

次のコードの一部は、リモート・プロシージャ・コール (RPC) で認証システムが作用する方法を表します。OS/400 に備わっている唯一の認証方法はシステムによるものです。次の情報が設定され、毎回の `clnt_call()` でクライアントからサービスに渡されます。次のコードの一部では、`rpc_call()` はデフォルトで `authnone` (空の認証トークン) を使用するので、認証情報を使用する際に `rpc_call()` では十分でないという点に注意してください。

- `aup_time` - 認証情報のタイム・スタンプ
- `aup_machname` - リモート・クライアントのホスト名
- `aup_uid` - クライアントのリモート・ユーザーの UID
- `aup_gid` - リモート・ユーザーの 1 次 GID
- `aup_gids` - リモート・ユーザーの 2 次グループの配列

認証情報は、リモート要求の一部としてサービスに直接入ります。この情報を解析し、クライアントが承認されたマシン、および承認されたユーザーからきているかどうかを検証することはサーバー側の責任です。認証タイプが間違っている場合、またはサーバーが受け入れられないほど不適切なものである場合、`svcerr_weakauth()` を使ってエラーが戻され、これをクライアントに通知します。

```

#include <sys/types.h> /* needed for gid_t and uid_t */
#include <stdlib.h> /* misc. system auth APIs */
#include <errno.h>

struct authsys_parms *credentials; /* authentication information */
char *remote_machine; /* machine name (from the credentials) */
uid_t remote_user; /* remote user's UID (from credentials) */

/* make sure we got the correct flavor of authentication */
if (request->rq_cred.oa_flavor != AUTH_UNIX) {
    /* if not, send back a weak authentication message and return */
    svcerr_weakauth(svc);
    return;
}

/* get our credentials */
credentials = (struct authsys_parms *) (request->rq_clntcred);

/* get the remote user's GID */
remote_user = credentials->aup_uid;

/* get the remote hostname of the client */
remote_machine = credentials->aup_machname;

/* check to see if this machine is "trusted" by us */
if ((strcmp("remote1", remote_machine) != 0) &&
    (strcmp("remote2", remote_machine) != 0)) {

    /* not from a machine we trust */
    /* send back an authentication error the client */

```

```

    svcerr_weakauth(svc);
    return;
} /* end of if (!trusted hostname) */

else {

    /* now check the user id for one we trust */
    /* information can be gotten from DSPUSRPRF */
    if ((remote_user != 568) &&
        (remote_user != 550) &&
        (remote_user != 528)) {

        /* not a user id we trust */
        /* send back an authentication error the client */
        svcerr_weakauth(svc);
        return;

    } /* end of if (!trusted uid) */
} /* end of else (trusted hostname) */

/* we fall out of the loop if the hostname and uid are trusted */

```

TI-RPC コード例に基づいてクライアント・アプリケーションを開発する

トランスポート層に依存しないリモート・プロシージャ・コール (TI-RPC) プログラミングは、OS/400 上でクライアント/サーバー間の分散アプリケーションを開発するための効率的な方法を提供します。TI-RPC サービスのアプリケーション・プログラミング・インターフェース (API) についての詳細は、System API Reference を参照してください。

OS/400 上でクライアント・アプリケーションを開発するには、以下のコード例をガイドラインとして使用してください。

- 『例: TI-RPC 簡易レベル・クライアント API』
- 22 ページの 『例: TI-RPC トップレベル・クライアント API』
- 26 ページの 『例: TI-RPC 中間レベル・クライアント API』
- 31 ページの 『例: TI-RPC エキスパート・レベル・クライアント API』
- 36 ページの 『例: TI-RPC クライアントへの認証の追加』

関連情報

- 1 ページの 『Sun TI-RPC を使用して分散アプリケーションを開発する』
- 4 ページの 『例: TI-RPC コードに基づいてサービス・アプリケーションを開発する』

例: TI-RPC 簡易レベル・クライアント API

次のコード例では、トランスポート層に依存しないリモート・プロシージャ・コール (TI-RPC) アプリケーションの開発に使用される簡易レベル・クライアント・アプリケーション・プログラミング・インターフェース (API) を例示します。

簡易レベル・クライアント API は、最も速く短いコードのセットです。これは、クライアント作成、制御、使用、および消滅がすべて 1 回のコールで済むためです。便利ではありますが、クライアント・ハンドルを使ってカスタマイズすることはできません。タイムアウトとバッファ・サイズについては、デフォルトが受け入れられます。これが簡易レベルと他のレベルの間の最も重要な相違点です。

```

#include <stdio.h>
#include <errno.h>
#include "myapp.h"

#define EXIT 100

int main(void) {

    enum clnt_stat rslt;    /* return value of rpc_call() */
    char hostname[256];    /* buffer for remote service's hostname */
    unsigned long procnum; /* procedure to call */
    char filename[512];    /* buffer for filename */
    char *arg = filename;  /* pointer to filename buffer */

    union {
        u_int    myapp_get_uid_result;
        char *   myapp_get_uid_string_result;
        int      myapp_get_size_result;
        long     myapp_get_mtime_result;
        char *   myapp_get_mtime_string_result;
        u_short  myapp_get_codepage_result;
        char *   myapp_get_objtype_result;
        char *   myapp_get_filetype_result;
    } result; /* a union of all the possible results */

    /* get the hostname from the user */
    printf("Enter the hostname where the remote service is running: %n");
    scanf("%s", (char *)&hostname);

    myapp_print_menu(); /* print out the menu choices */

    /* get the procedure number to call from the user */
    printf("%nEnter a procedure number to call: %n");
    scanf("%lu", &procnum);

    /* get the filename from the user */
    printf("%nEnter a filename to stat: %n");
    scanf("%s", (char *)&arg);

    /* switch on the input */
    switch (procnum) {

        case NULLPROC:

            /* rpc_call(host, prognum, versnum, procnum,
            /*      xdr_in, in, xdr_out, out, nettype); */
            rslt = rpc_call(hostname, PROGNUM, VERSNUM, procnum,
                (xdrproc_t)xdr_void, (char *)NULL, /* xdr_in */
                (xdrproc_t)xdr_void, (char *)NULL, /* xdr_out */
                NETTYPE);

            /* check return value of rpc_call() */
            if (rslt != RPC_SUCCESS) {
                fprintf(stderr, "Error calling rpc_call(%lu)%n", procnum);
                fprintf(stderr, "clnt_stat: %d%n", rslt);
                fprintf(stderr, "errno: %d%n", errno);
                return 1;
            }

            /* print results and exit */
            printf("NULLRPOC call succeeded%n");
            break;

        case GET_UID:

            /* rpc_call(host, prognum, versnum, procnum,
            /*      xdr_in, in, xdr_out, out, nettype); */

```



```

rslt = rpc_call(hostname, PROGNUM, VERSNUM, procnum,
                xdr_wrapstring, (char *)&arg, /* xdr_in */
                xdr_u_int, (char *)&result, /* xdr_out */
                NETTYPE);

/* check return value of rpc_call() */
if (rslt != RPC_SUCCESS) {
    fprintf(stderr, "Error calling rpc_call(%lu)%n", procnum);
    fprintf(stderr, "cInt_stat: %d%n", rslt);
    fprintf(stderr, "errno: %d%n", errno);
    return 1;
}

/* print results and exit */
printf("uid of %s: %u%n",
       filename, result.myapp_get_uid_result);
break;

case GET_UID_STRING:

/* rpc_call(host, prognum, versnum, procnum, /*
/*      xdr_in, in, xdr_out, out, nettype); */
rslt = rpc_call(hostname, PROGNUM, VERSNUM, procnum,
                xdr_wrapstring, (char *)&arg, /* xdr_in */
                xdr_wrapstring, (char *)&result, /* xdr_out */
                NETTYPE);

/* check return value of rpc_call() */
if (rslt != RPC_SUCCESS) {
    fprintf(stderr, "Error calling rpc_call(%lu)%n", procnum);
    fprintf(stderr, "cInt_stat: %d%n", rslt);
    fprintf(stderr, "errno: %d%n", errno);
    return 1;
}

/* print results and exit */
printf("owner of %s: %s%n",
       filename, result.myapp_get_uid_string_result);
break;

case GET_SIZE:

/* rpc_call(host, prognum, versnum, procnum, /*
/*      xdr_in, in, xdr_out, out, nettype); */
rslt = rpc_call(hostname, PROGNUM, VERSNUM, procnum,
                xdr_wrapstring, (char *)&arg, /* xdr_in */
                xdr_int, (char *)&result, /* xdr_out */
                NETTYPE);

/* check return value of rpc_call() */
if (rslt != RPC_SUCCESS) {
    fprintf(stderr, "Error calling rpc_call(%lu)%n", procnum);
    fprintf(stderr, "cInt_stat: %d%n", rslt);
    fprintf(stderr, "errno: %d%n", errno);
    return 1;
}

/* print results and exit */
printf("size of %s: %d%n",
       filename, result.myapp_get_size_result);
break;

case GET_MTIME:

/* rpc_call(host, prognum, versnum, procnum, /*
/*      xdr_in, in, xdr_out, out, nettype); */
rslt = rpc_call(hostname, PROGNUM, VERSNUM, procnum,

```

```

        xdr_wrapstring, (char *)&arg, /* xdr_in */
        xdr_long, (char *)&result, /* xdr_out */
        NETTYPE);

/* check return value of rpc_call() */
if (rslt != RPC_SUCCESS) {
    fprintf(stderr, "Error calling rpc_call(%lu)%n", procnum);
    fprintf(stderr, "clnt_stat: %d%n", rslt);
    fprintf(stderr, "errno: %d%n", errno);
    return 1;
}

/* print results and exit */
printf("last modified time of %s: %ld%n",
       filename, result.myapp_get_mtime_result);
break;

case GET_MTIME_STRING:

/* rpc_call(host, prognum, versnum, procnum,
/*      xdr_in, in, xdr_out, out, nettype); */
rslt = rpc_call(hostname, PROGNUM, VERSNUM, procnum,
                xdr_wrapstring, (char *)&arg, /* xdr_in */
                xdr_wrapstring, (char *)&result, /* xdr_out */
                NETTYPE);

/* check return value of rpc_call() */
if (rslt != RPC_SUCCESS) {
    fprintf(stderr, "Error calling rpc_call(%lu)%n", procnum);
    fprintf(stderr, "clnt_stat: %d%n", rslt);
    fprintf(stderr, "errno: %d%n", errno);
    return 1;
}

/* print results and exit */
printf("last modified time of %s: %s%n",
       filename, result.myapp_get_mtime_string_result);
break;

case GET_CODEPAGE:

/* rpc_call(host, prognum, versnum, procnum,
/*      xdr_in, in, xdr_out, out, nettype); */
rslt = rpc_call(hostname, PROGNUM, VERSNUM, procnum,
                xdr_wrapstring, (char *)&arg, /* xdr_in */
                xdr_u_short, (char *)&result, /* xdr_out */
                NETTYPE);

/* check return value of rpc_call() */
if (rslt != RPC_SUCCESS) {
    fprintf(stderr, "Error calling rpc_call(%lu)%n", procnum);
    fprintf(stderr, "clnt_stat: %d%n", rslt);
    fprintf(stderr, "errno: %d%n", errno);
    return 1;
}

/* print results and exit */
printf("codepage of %s: %d%n",
       filename, result.myapp_get_codepage_result);
break;

case GET_OBJTYPE:

/* rpc_call(host, prognum, versnum, procnum,
/*      xdr_in, in, xdr_out, out, nettype); */
rslt = rpc_call(hostname, PROGNUM, VERSNUM, procnum,
                xdr_wrapstring, (char *)&arg, /* xdr_in */

```

```

        xdr_wrapstring, (char *)&result, /* xdr_out */
        NETTYPE);

/* check return value of rpc_call() */
if (rslt != RPC_SUCCESS) {
    fprintf(stderr, "Error calling rpc_call(%lu)%n", procnum);
    fprintf(stderr, "clnt_stat: %d%n", rslt);
    fprintf(stderr, "errno: %d%n", errno);
    return 1;
}

/* print results and exit */
printf("object type of %s: %s%n",
       filename, result.myapp_get_objtype_result);
break;

case GET_FILETYPE:

/* rpc_call(host, prognum, versnum, procnum,
/*      xdr_in, in, xdr_out, out, nettype); */
rslt = rpc_call(hostname, PROGNUM, VERSNUM, procnum,
                xdr_wrapstring, (char *)&arg, /* xdr_in */
                xdr_wrapstring, (char *)&result, /* xdr_out */
                NETTYPE);

/* check return value of rpc_call() */
if (rslt != RPC_SUCCESS) {
    fprintf(stderr, "Error calling rpc_call(%lu)%n", procnum);
    fprintf(stderr, "clnt_stat: %d%n", rslt);
    fprintf(stderr, "errno: %d%n", errno);
    return 1;
}

/* print results and exit */
printf("file type of %s: %s%n",
       filename, result.myapp_get_filetype_result);
break;

case END_SERVER:

/* rpc_call(host, prognum, versnum, procnum,
/*      xdr_in, in, xdr_out, out, nettype); */
rslt = rpc_call(hostname, PROGNUM, VERSNUM, procnum,
                (xdrproc_t)xdr_void, (char *)NULL, /* xdr_in */
                (xdrproc_t)xdr_void, (char *)NULL, /* xdr_out */
                NETTYPE);

/* check return value of rpc_call() */
if (rslt != RPC_SUCCESS) {
    fprintf(stderr, "Error calling rpc_call(%lu)%n", procnum);
    fprintf(stderr, "clnt_stat: %d%n", rslt);
    fprintf(stderr, "errno: %d%n", errno);
    return 1;
}

/* print results and exit */
printf("Service has been unregistered.%n");
printf("You must still kill the job in QBATCH%n");
break;

case EXIT:

/* do nothing and exit */
printf("Exiting program now.%n");
return 1;
break;

```

```

default:

    /* an invalid procedure number was entered */
    /* we could just exit here */
    printf("Invalid choice. Issuing NULLRPOC instead.\n");
    procnum = NULLPROC;

    /* rpc_call(host, prognum, versnum, procnum,
    /*      xdr_in, in, xdr_out, out, nettype); */
    rslt = rpc_call(hostname, PROGNUM, VERSNUM, procnum,
                    (xdrproc_t)xdr_void, (char *)NULL, /* xdr_in */
                    (xdrproc_t)xdr_void, (char *)NULL, /* xdr_out */
                    NETTYPE);

    /* check return value of rpc_call() */
    if (rslt != RPC_SUCCESS) {
        fprintf(stderr, "Error calling rpc_call(%lu)\n", procnum);
        fprintf(stderr, "clnt_stat: %d\n", rslt);
        fprintf(stderr, "errno: %d\n", errno);
        return 1;
    }

    /* print results and exit */
    printf("NULLRPOC call succeeded\n");
    break;

} /* end of switch(procnum) */

/* no cleanup is required for rpc_call() */
return 0;

}

void myapp_print_menu(void) {

    /* print out the procedure choices */
    printf("%.21d - GET_UID          %.21d - GET_UID_STRING\n",
           GET_UID, GET_UID_STRING);
    printf("%.21d - GET_SIZE        %.21d - GET_MTIME\n",
           GET_SIZE, GET_MTIME);
    printf("%.21d - GET_MTIME_STRING %.21d - GET_CODEPAGE\n",
           GET_MTIME_STRING, GET_CODEPAGE);
    printf("%.21d - GET_OBJTYPE     %.21d - GET_FILETYPE\n",
           GET_OBJTYPE, GET_FILETYPE);
    printf("%.21d - END_SERVER      %.2d - EXIT\n",
           END_SERVER, EXIT);

}

```

例: TI-RPC トップレベル・クライアント API

次のコード例では、トランスポート層に依存しないリモート・プロシージャ・コール (TI-RPC) アプリケーションの開発に使用されるトップレベル・クライアント・アプリケーション・プログラミング・インターフェース (API) を例示します。

トップレベルでは、使用または修正する前にクライアント・ハンドルを作成することが必要です。トップレベル API は簡単に使用でき、さらには簡易レベルよりも多くの操作とエラー処理が可能です。

```

#include <stdio.h>
#include <netconfig.h>
#include <netdir.h>
#include <errno.h>
#include "myapp.h"

#define EXIT 100

```

```

int main(void) {

enum clnt_stat rslt; /* return value of clnt_call() */
char hostname[256]; /* buffer for remote service's hostname */
unsigned long procnum; /* procedure to call */
char filename[512]; /* buffer for filename */
xdrproc_t xdr_argument; /* xdr procedure to encode arguments */
xdrproc_t xdr_result; /* xdr procedure to decode results */
CLIENT *clnt; /* pointer to client handle */
struct timeval tout; /* timeout for clnt_call() */
char *arg = filename; /* pointer to filename buffer */

union {
    u_int myapp_get_uid_result;
    char * myapp_get_uid_string_result;
    int myapp_get_size_result;
    long myapp_get_mtime_result;
    char * myapp_get_mtime_string_result;
    u_short myapp_get_codepage_result;
    char * myapp_get_objtype_result;
    char * myapp_get_filetype_result;
} result; /* a union of all the possible results */

tout.tv_sec = 30; /* set default timeout to 30.00 seconds */
tout.tv_usec = 0;

/* get the hostname from the user */
printf("Enter the hostname where the remote service is running: %n");
scanf("%s", (char *)&hostname);

myapp_print_menu(); /* print out the menu choices */

/* get the procedure number to call from the user */
printf("%nEnter a procedure number to call: %n");
scanf("%lu", &procnum);

/* get the filename from the user */
printf("%nEnter a filename to stat: %n");
scanf("%s", (char *)&filename);

/* clnt_create(host, prognum, versnum, nettype); */
clnt = clnt_create(hostname, PROGNUM, VERSNUM, NETTYPE);

/* check to make sure clnt_create() didn't fail */
if (clnt == (CLIENT *)NULL) {
    /* if we failed, print out all appropriate error messages and exit */
    fprintf(stderr, "Error calling clnt_create(%n");
    fprintf(stderr, "PROG: %lu%tVERS: %lu%tNET: %s%n",
            PROGNUM, VERSNUM, NETTYPE);
    fprintf(stderr, "clnt_stat: %d%n", rpc_createerr.cf_stat);
    fprintf(stderr, "errno: %d%n", errno);
    fprintf(stderr, "re_errno: %d%n", rpc_createerr.cf_error.re_errno);
    return 1;
}

/* switch on the input */
switch (procnum) {

    case NULLPROC:
        /* set the encode procedure */
        xdr_argument = (xdrproc_t)xdr_void;
        /* set the decode procedure */
        xdr_result = (xdrproc_t)xdr_void;
        break;

    case GET_UID:

```

```

    /* set the encode procedure */
    xdr_argument = xdr_wrapstring;
    /* set the decode procedure */
    xdr_result = xdr_u_int;
    break;

case GET_UID_STRING:
    /* set the encode procedure */
    xdr_argument = xdr_wrapstring;
    /* set the decode procedure */
    xdr_result = xdr_wrapstring;
    break;

case GET_SIZE:
    /* set the encode procedure */
    xdr_argument = xdr_wrapstring;
    /* set the decode procedure */
    xdr_result = xdr_int;
    break;

case GET_MTIME:
    /* set the encode procedure */
    xdr_argument = xdr_wrapstring;
    /* set the decode procedure */
    xdr_result = xdr_long;
    break;

case GET_MTIME_STRING:
    /* set the encode procedure */
    xdr_argument = xdr_wrapstring;
    /* set the decode procedure */
    xdr_result = xdr_wrapstring;
    break;

case GET_CODEPAGE:
    /* set the encode procedure */
    xdr_argument = xdr_wrapstring;
    /* set the decode procedure */
    xdr_result = xdr_u_short;
    break;

case GET_OBJTYPE:
    /* set the encode procedure */
    xdr_argument = xdr_wrapstring;
    /* set the decode procedure */
    xdr_result = xdr_wrapstring;
    break;

case GET_FILETYPE:
    /* set the encode procedure */
    xdr_argument = xdr_wrapstring;
    /* set the decode procedure */
    xdr_result = xdr_wrapstring;
    break;

case END_SERVER:
    /* set the encode procedure */
    xdr_argument = (xdrproc_t)xdr_void;
    /* set the decode procedure */
    xdr_result = (xdrproc_t)xdr_void;
    break;

case EXIT:
    /* we're done. clean up and exit */
    clnt_destroy(clnt);
    return 1;
    break;

```

```

default:
    /* invalid procedure number entered. defaulting to NULLPROC */
    printf("Invalid choice. Issuing NULLRPOC instead.\n");
    procnum = NULLPROC;
    /* set the encode procedure */
    xdr_argument = (xdrproc_t)xdr_void;
    /* set the decode procedure */
    xdr_result = (xdrproc_t)xdr_void;
    break;
} /* end of switch(procnum) */

/* clnt_call(client, procnum, xdr_inproc, in, xdr_outproc, out, timeout) */
rslt = clnt_call(clnt, procnum, xdr_argument, (char *)&arg,
                xdr_result, (char *)&result, tout);

/* check to make sure clnt_call() succeeded */
if (rslt != RPC_SUCCESS) {
    /* if clnt_call() failed, print errors and exit */
    printf("An error occurred calling %lu procedure\n", procnum);
    printf("clnt_stat: %d\terrno: %d\n", rslt, errno);
    clnt_destroy(clnt);
    return 1;
}

/* clnt_call() succeeded. switch on procedure and print results */
switch (procnum) {

    case NULLPROC:
        /* print results and exit */
        printf("NULLRPOC call succeeded\n");
        break;

    case GET_UID:
        /* print results and exit */
        printf("uid of %s: %u\n",
              filename, result.myapp_get_uid_result);
        break;

    case GET_UID_STRING:
        /* print results and exit */
        printf("owner of %s: %s\n",
              filename, result.myapp_get_uid_string_result);
        break;

    case GET_SIZE:
        /* print results and exit */
        printf("size of %s: %d\n",
              filename, result.myapp_get_size_result);
        break;

    case GET_MTIME:
        /* print results and exit */
        printf("last modified time of %s: %ld\n",
              filename, result.myapp_get_mtime_result);
        break;

    case GET_MTIME_STRING:
        /* print results and exit */
        printf("last modified time of %s: %s\n",
              filename, result.myapp_get_mtime_string_result);
        break;

    case GET_CODEPAGE:
        /* print results and exit */
        printf("codepage of %s: %d\n",

```

```

        filename, result.myapp_get_codepage_result);
    break;

case GET_OBJTYPE:
    /* print results and exit */
    printf("object type of %s: %s\n",
        filename, result.myapp_get_objtype_result);
    break;

case GET_FILETYPE:
    /* print results and exit */
    printf("file type of %s: %s\n",
        filename, result.myapp_get_filetype_result);
    break;

case END_SERVER:
    /* print results and exit */
    printf("Service has been unregistered.\n");
    printf("You must still kill the job in QBATCH\n");
    break;

default:
    /* we should never get the default case. */
    /* the previous switch should catch it. */
    break;

} /* end of switch(procnum) */

/* clean up and exit */
clnt_destroy(clnt);

return 0;
}

void myapp_print_menu(void) {

    /* print out the procedure choices */
    printf("%.21d - GET_UID          %.21d - GET_UID_STRING\n",
        GET_UID, GET_UID_STRING);
    printf("%.21d - GET_SIZE        %.21d - GET_MTIME\n",
        GET_SIZE, GET_MTIME);
    printf("%.21d - GET_MTIME_STRING  %.21d - GET_CODEPAGE\n",
        GET_MTIME_STRING, GET_CODEPAGE);
    printf("%.21d - GET_OBJTYPE      %.21d - GET_FILETYPE\n",
        GET_OBJTYPE, GET_FILETYPE);
    printf("%.21d - END_SERVER      %.2d - EXIT\n",
        END_SERVER, EXIT);
}

```

例: TI-RPC 中間レベル・クライアント API

次のコード例では、トランスポート層に依存しないリモート・プロシージャ・コール (TI-RPC) アプリケーションの開発に使用される中間レベル・クライアント・アプリケーション・プログラミング・インターフェース (API) を例示します。

クライアントの中間レベルは、サービスの中間レベルと同じパスに従います。たとえば、プログラマーは簡単なテキスト・ストリングを渡すのではなく、ネットワーク選択 API を使ってトランスポート情報を入手する責任があります。

```

#include <stdio.h>
#include <netconfig.h>
#include <netdir.h>

```



```

#include <errno.h>
#include "myapp.h"

#define EXIT 100

int main(void) {

    enum clnt_stat rslt; /* return value of clnt_call() */
    char hostname[256]; /* buffer for remote service's hostname */
    unsigned long procnum; /* procedure to call */
    char filename[512]; /* buffer for filename */
    xdrproc_t xdr_argument; /* xdr procedure to encode arguments */
    xdrproc_t xdr_result; /* xdr procedure to decode results */
    CLIENT *clnt; /* pointer to client handle */
    struct timeval tout; /* timeout for clnt_call() */
    struct netconfig *nconf; /* transport information */
    char *arg = filename; /* pointer to filename buffer */

    union {
        u_int myapp_get_uid_result;
        char * myapp_get_uid_string_result;
        int myapp_get_size_result;
        long myapp_get_mtime_result;
        char * myapp_get_mtime_string_result;
        u_short myapp_get_codepage_result;
        char * myapp_get_objtype_result;
        char * myapp_get_filetype_result;
    } result; /* a union of all the possible results */

    tout.tv_sec = 30; /* set default timeout to 30.00 seconds */
    tout.tv_usec = 0;

    /* get the hostname from the user */
    printf("Enter the hostname where the remote service is running: %n");
    scanf("%s", (char *)&hostname);

    myapp_print_menu(); /* print out the menu choices */

    /* get the procedure number to call from the user */
    printf("%nEnter a procedure number to call: %n");
    scanf("%lu", &procnum);

    /* get the filename from the user */
    printf("%nEnter a filename to stat: %n");
    scanf("%s", (char *)&filename);

    /* getnetconfigent(nettype) */
    nconf = getnetconfigent(NETTYPE);

    /* check to make sure getnetconfigent() didn't fail */
    if (nconf == NULL) {
        /* if getnetconfigent() failed, print error messages and exit */
        fprintf(stderr, "Error calling getnetconfigent(%s)%n", NETTYPE);
        fprintf(stderr, "errno: %d%n", errno);
        return 1;
    }

    /* clnt_tp_create(host, prognum, versnum, netconf) */
    clnt = clnt_tp_create(hostname, PROGNUM, VERSNUM, nconf);

    /* check to make sure clnt_tp_create() didn't fail */
    if (clnt == (CLIENT *)NULL) {
        fprintf(stderr, "Error calling clnt_tp_create()%n");
        fprintf(stderr, "PROG: %lu%tVERS: %lu%tNET: %s%n",
            PROGNUM, VERSNUM, NETTYPE);
        fprintf(stderr, "clnt_stat: %d%n", rpc_createerr.cf_stat);
    }
}

```

```

    fprintf(stderr, "errno: %d\n", errno);
    fprintf(stderr, "re_errno: %d\n", rpc_createerr.cf_error.re_errno);
    return 1;
}

/* switch on the input */
switch (procnum) {

    case NULLPROC:
        /* set the encode procedure */
        xdr_argument = (xdrproc_t)xdr_void;
        /* set the decode procedure */
        xdr_result = (xdrproc_t)xdr_void;
        break;

    case GET_UID:
        /* set the encode procedure */
        xdr_argument = xdr_wrapstring;
        /* set the decode procedure */
        xdr_result = xdr_u_int;
        break;

    case GET_UID_STRING:
        /* set the encode procedure */
        xdr_argument = xdr_wrapstring;
        /* set the decode procedure */
        xdr_result = xdr_wrapstring;
        break;

    case GET_SIZE:
        /* set the encode procedure */
        xdr_argument = xdr_wrapstring;
        /* set the decode procedure */
        xdr_result = xdr_int;
        break;

    case GET_MTIME:
        /* set the encode procedure */
        xdr_argument = xdr_wrapstring;
        /* set the decode procedure */
        xdr_result = xdr_long;
        break;

    case GET_MTIME_STRING:
        /* set the encode procedure */
        xdr_argument = xdr_wrapstring;
        /* set the decode procedure */
        xdr_result = xdr_wrapstring;
        break;

    case GET_CODEPAGE:
        /* set the encode procedure */
        xdr_argument = xdr_wrapstring;
        /* set the decode procedure */
        xdr_result = xdr_u_short;
        break;

    case GET_OBJTYPE:
        /* set the encode procedure */
        xdr_argument = xdr_wrapstring;
        /* set the decode procedure */
        xdr_result = xdr_wrapstring;
        break;

    case GET_FILETYPE:
        /* set the encode procedure */
        xdr_argument = xdr_wrapstring;

```

```

    /* set the decode procedure */
    xdr_result = xdr_wrapstring;
    break;

case END_SERVER:
    /* set the encode procedure */
    xdr_argument = (xdrproc_t)xdr_void;
    /* set the decode procedure */
    xdr_result = (xdrproc_t)xdr_void;
    break;

case EXIT:
    /* we're done. clean up and exit */
    clnt_destroy(clnt);
    return 1;
    break;

default:
    /* invalid procedure number entered. defaulting to NULLPROC */
    printf("Invalid choice. Issuing NULLRPOC instead.\n");
    procnum = NULLPROC;
    /* set the encode procedure */
    xdr_argument = (xdrproc_t)xdr_void;
    /* set the decode procedure */
    xdr_result = (xdrproc_t)xdr_void;
    break;
} /* end of switch(procnum) */

/* clnt_call(client, procnum, xdr_inproc, in, xdr_outproc, out, timeout) */
rslt = clnt_call(clnt, procnum, xdr_argument, (char *)&arg,
                xdr_result, (char *)&result, tout);

/* check to make sure clnt_call() succeeded */
if (rslt != RPC_SUCCESS) {
    /* if clnt_call() failed, print errors and exit */
    printf("An error occurred calling %lu procedure\n", procnum);
    printf("clnt_stat: %d\terrno: %d\n", rslt, errno);
    clnt_destroy(clnt);
    return 1;
}

/* clnt_call() succeeded. switch on procedure and print results */
switch (procnum) {

case NULLPROC:
    /* print results and exit */
    printf("NULLRPOC call succeeded\n");
    break;

case GET_UID:
    /* print results and exit */
    printf("uid of %s: %u\n",
           filename, result.myapp_get_uid_result);
    break;

case GET_UID_STRING:
    /* print results and exit */
    printf("owner of %s: %s\n",
           filename, result.myapp_get_uid_string_result);
    break;

case GET_SIZE:
    /* print results and exit */
    printf("size of %s: %d\n",
           filename, result.myapp_get_size_result);
    break;
}

```

```

case GET_MTIME:
    /* print results and exit */
    printf("last modified time of %s: %ld\n",
           filename, result.myapp_get_mtime_result);
    break;

case GET_MTIME_STRING:
    /* print results and exit */
    printf("last modified time of %s: %s\n",
           filename, result.myapp_get_mtime_string_result);
    break;

case GET_CODEPAGE:
    /* print results and exit */
    printf("codepage of %s: %d\n",
           filename, result.myapp_get_codepage_result);
    break;

case GET_OBJTYPE:
    /* print results and exit */
    printf("object type of %s: %s\n",
           filename, result.myapp_get_objtype_result);
    break;

case GET_FILETYPE:
    /* print results and exit */
    printf("file type of %s: %s\n",
           filename, result.myapp_get_filetype_result);
    break;

case END_SERVER:
    /* print results and exit */
    printf("Service has been unregistered.\n");
    printf("You must still kill the job in QBATCH\n");
    break;

default:
    /* we should never get the default case. */
    /* the previous switch should catch it. */
    break;

} /* end of switch(procnum) */

/* clean up and exit */

/* free the netconfig struct */
freenetconfigent(nconf);
/* free the universal address buffer */
free(svcaddr.buf);
/* destroy the client handle */
clnt_destroy(clnt);

return 0;
}

void myapp_print_menu(void) {
    /* print out the procedure choices */
    printf("%.21d - GET_UID          %.21d - GET_UID_STRING\n",
           GET_UID, GET_UID_STRING);
    printf("%.21d - GET_SIZE        %.21d - GET_MTIME\n",
           GET_SIZE, GET_MTIME);
    printf("%.21d - GET_MTIME_STRING %.21d - GET_CODEPAGE\n",

```

```

        GET_MTIME_STRING, GET_CODEPAGE);
printf("%.21d - GET_OBJTYPE      %.21d - GET_FILETYPE\n",
        GET_OBJTYPE, GET_FILETYPE);
printf("%.21d - END_SERVER      %.2d - EXIT\n",
        END_SERVER, EXIT);
}

```

例: TI-RPC エキスパート・レベル・クライアント API

次のコード例では、トランスポート層に依存しないリモート・プロシージャ・コール (TI-RPC) アプリケーションの開発に使用されるエキスパート・レベル・クライアント・アプリケーション・プログラミング・インターフェース (API) を例示します。

クライアント API の開発用のエキスパート・レベルはより複雑です。また、カスタマイズを最大限に行えます。これは、クライアント API に合わせてバッファ・サイズを調整できる唯一のレベルです。エキスパート・レベルでは、プログラマーは名前からアドレスへの変換 API または他のエキスパート・レベル API のどちらかを使用することによって、クライアントが接続するための汎用アドレスを設定することが必要です。どちらにしても多くの作業が必要になりますが、プログラマーは、このレベルが実行する環境に合わせてクライアント・アプリケーションを調整することが可能になります。

```

#include <stdio.h>
#include <netconfig.h>
#include <netdir.h>
#include <errno.h>
#include "myapp.h"

#define EXIT 100

int main(void) {

    enum clnt_stat rslt; /* return value of clnt_call() */
    char hostname[256]; /* buffer for remote service's hostname */
    unsigned long procnum; /* procedure to call */
    char filename[512]; /* buffer for filename */
    xdrproc_t xdr_argument; /* xdr procedure to encode arguments */
    xdrproc_t xdr_result; /* xdr procedure to decode results */
    CLIENT *clnt; /* pointer to client handle */
    struct timeval tout; /* timeout for clnt_call() */
    struct netconfig *nconf; /* transport information */
    struct netbuf svcaddr; /* universal address of remote service */
    bool_t rpcb_rslt; /* return value for rpcb_getaddr() */
    char *arg = filename; /* pointer to filename buffer */

    union {
        u_int myapp_get_uid_result;
        char * myapp_get_uid_string_result;
        int myapp_get_size_result;
        long myapp_get_mtime_result;
        char * myapp_get_mtime_string_result;
        u_short myapp_get_codepage_result;
        char * myapp_get_objtype_result;
        char * myapp_get_filetype_result;
    } result; /* a union of all the possible results */

    /* initialize the struct netbuf space */
    svcaddr.maxlen = 16;
    svcaddr.buf = (char *)malloc(svcaddr.maxlen);

    if (svcaddr.buf == (char *)NULL) {
        /* if malloc() failed, print error messages and exit */
        fprintf(stderr, "Error calling malloc() for struct netbuf\n");
        fprintf(stderr, "errno: %d\n", errno);
        return 1;
    }
}

```

```

}

tout.tv_sec = 30; /* set default timeout to 30.00 seconds */
tout.tv_usec = 0;

/* get the hostname from the user */
printf("Enter the hostname where the remote service is running: %n");
scanf("%s", (char *)&hostname);

myapp_print_menu(); /* print out the menu choices */

/* get the procedure number to call from the user */
printf("%nEnter a procedure number to call: %n");
scanf("%lu", &procnum);

/* get the filename from the user */
printf("%nEnter a filename to stat: %n");
scanf("%s", (char *)&filename);

/* getnetconfigent(nettype) */
nconf = getnetconfigent(NETTYPE);

/* check to make sure getnetconfigent() didn't fail */
if (nconf == NULL) {
    /* if getnetconfigent() failed, print error messages and exit */
    fprintf(stderr, "Error calling getnetconfigent(%s)%n", NETTYPE);
    fprintf(stderr, "errno: %d%n", errno);
    return 1;
}

/* rpcb_getaddr(prognum, versnum, nconf, output netbuf, hostname) */
/* this sets the universal address svcaddr */
rpcb_rslt = rpcb_getaddr(PROGNUM, VERSNUM, nconf, &svcaddr, hostname);

/* check to make sure rpcb_getaddr() didn't fail */
if (rpcb_rslt == FALSE) {
    /* if rpcb_getaddr() failed, print error messages and exit */
    fprintf(stderr, "Error calling rpcb_getaddr()%n");
    fprintf(stderr, "PROG: %lu%tVERS: %lu%tNET: %s%n",
        PROGNUM, VERSNUM, NETTYPE);
    fprintf(stderr, "clnt_stat: %d%n", rpc_createerr.cf_stat);
    fprintf(stderr, "errno: %d%n", errno);
    fprintf(stderr, "re_errno: %d%n", rpc_createerr.cf_error.re_errno);
    return 1;
}

/* clnt_tli_create(filedes, netconfig, netbuf,          */
/*                  prognum, versnum, sendsz, recvsz); */
clnt = clnt_tli_create(RPC_ANYFD, nconf, &svcaddr,
    PROGNUM, VERSNUM, 0, 0);

/* check to make sure clnt_tli_create() didn't fail */
if (clnt == (CLIENT *)NULL) {
    /* if we failed, print out all appropriate error messages and exit */
    fprintf(stderr, "Error calling clnt_tli_create()%n");
    fprintf(stderr, "PROG: %lu%tVERS: %lu%tNET: %s%n",
        PROGNUM, VERSNUM, NETTYPE);
    fprintf(stderr, "clnt_stat: %d%n", rpc_createerr.cf_stat);
    fprintf(stderr, "errno: %d%n", errno);
    fprintf(stderr, "re_errno: %d%n", rpc_createerr.cf_error.re_errno);
    return 1;
}

/* switch on the input */
switch (procnum) {

    case NULLPROC:

```

```

    /* set the encode procedure */
    xdr_argument = (xdrproc_t)xdr_void;
    /* set the decode procedure */
    xdr_result = (xdrproc_t)xdr_void;
    break;

case GET_UID:
    /* set the encode procedure */
    xdr_argument = xdr_wrapstring;
    /* set the decode procedure */
    xdr_result = xdr_u_int;
    break;

case GET_UID_STRING:
    /* set the encode procedure */
    xdr_argument = xdr_wrapstring;
    /* set the decode procedure */
    xdr_result = xdr_wrapstring;
    break;

case GET_SIZE:
    /* set the encode procedure */
    xdr_argument = xdr_wrapstring;
    /* set the decode procedure */
    xdr_result = xdr_int;
    break;

case GET_MTIME:
    /* set the encode procedure */
    xdr_argument = xdr_wrapstring;
    /* set the decode procedure */
    xdr_result = xdr_long;
    break;

case GET_MTIME_STRING:
    /* set the encode procedure */
    xdr_argument = xdr_wrapstring;
    /* set the decode procedure */
    xdr_result = xdr_wrapstring;
    break;

case GET_CODEPAGE:
    /* set the encode procedure */
    xdr_argument = xdr_wrapstring;
    /* set the decode procedure */
    xdr_result = xdr_u_short;
    break;

case GET_OBJTYPE:
    /* set the encode procedure */
    xdr_argument = xdr_wrapstring;
    /* set the decode procedure */
    xdr_result = xdr_wrapstring;
    break;

case GET_FILETYPE:
    /* set the encode procedure */
    xdr_argument = xdr_wrapstring;
    /* set the decode procedure */
    xdr_result = xdr_wrapstring;
    break;

case END_SERVER:
    /* set the encode procedure */
    xdr_argument = (xdrproc_t)xdr_void;
    /* set the decode procedure */
    xdr_result = (xdrproc_t)xdr_void;

```

```

        break;

case EXIT:
    /* we're done. clean up and exit */
    clnt_destroy(clnt);
    return 1;
    break;

default:
    /* invalid procedure number entered. defaulting to NULLPROC */
    printf("Invalid choice. Issuing NULLRPOC instead.¥n");
    procnum = NULLPROC;
    /* set the encode procedure */
    xdr_argument = (xdrproc_t)xdr_void;
    /* set the decode procedure */
    xdr_result = (xdrproc_t)xdr_void;
    break;
} /* end of switch(procnum) */

/* clnt_call(client, procnum, xdr_inproc, in, xdr_outproc, out, timeout) */
rslt = clnt_call(clnt, procnum, xdr_argument, (char *)&arg,
                xdr_result, (char *)&result, tout);

/* check to make sure clnt_call() succeeded */
if (rslt != RPC_SUCCESS) {
    /* if clnt_call() failed, print errors and exit */
    printf("An error occurred calling %lu procedure¥n", procnum);
    printf("clnt_stat: %d¥terrno: %d¥n", rslt, errno);
    clnt_destroy(clnt);
    return 1;
}

/* clnt_call() succeeded. switch on procedure and print results */
switch (procnum) {

case NULLPROC:
    /* print results and exit */
    printf("NULLRPOC call succeeded¥n");
    break;

case GET_UID:
    /* print results and exit */
    printf("uid of %s: %u¥n",
           filename, result.myapp_get_uid_result);
    break;

case GET_UID_STRING:
    /* print results and exit */
    printf("owner of %s: %s¥n",
           filename, result.myapp_get_uid_string_result);
    break;

case GET_SIZE:
    /* print results and exit */
    printf("size of %s: %d¥n",
           filename, result.myapp_get_size_result);
    break;

case GET_MTIME:
    /* print results and exit */
    printf("last modified time of %s: %ld¥n",
           filename, result.myapp_get_mtime_result);
    break;

case GET_MTIME_STRING:
    /* print results and exit */

```



```

        printf("last modified time of %s: %s\n",
               filename, result.myapp_get_mtime_string_result);
        break;

    case GET_CODEPAGE:
        /* print results and exit */
        printf("codepage of %s: %d\n",
               filename, result.myapp_get_codepage_result);
        break;

    case GET_OBJTYPE:
        /* print results and exit */
        printf("object type of %s: %s\n",
               filename, result.myapp_get_objtype_result);
        break;

    case GET_FILETYPE:
        /* print results and exit */
        printf("file type of %s: %s\n",
               filename, result.myapp_get_filetype_result);
        break;

    case END_SERVER:
        /* print results and exit */
        printf("Service has been unregistered.\n");
        printf("You must still kill the job in QBATCH\n");
        break;

    default:
        /* we should never get the default case. */
        /* the previous switch should catch it. */
        break;

} /* end of switch(procnum) */

/* clean up and exit */

/* free the netconfig struct */
freenetconfigent(nconf);
/* free the universal address buffer */
free(svcaddr.buf);
/* destroy the client handle */
clnt_destroy(clnt);

return 0;
}

void myapp_print_menu(void) {
    /* print out the procedure choices */
    printf("%.21d - GET_UID          %.21d - GET_UID_STRING\n",
           GET_UID, GET_UID_STRING);
    printf("%.21d - GET_SIZE          %.21d - GET_MTIME\n",
           GET_SIZE, GET_MTIME);
    printf("%.21d - GET_MTIME_STRING  %.21d - GET_CODEPAGE\n",
           GET_MTIME_STRING, GET_CODEPAGE);
    printf("%.21d - GET_OBJTYPE       %.21d - GET_FILETYPE\n",
           GET_OBJTYPE, GET_FILETYPE);
    printf("%.21d - END_SERVER        %.2d - EXIT\n",
           END_SERVER, EXIT);
}

```

例: TI-RPC クライアントへの認証の追加

次のコードの一部は、リモート・プロシージャ・コール (RPC) で認証システムが作用する方法を表します。OS/400 に備わっている唯一の認証方法はシステムによるものです。次の情報が設定され、毎回の `clnt_call()` でクライアントからサービスに渡されます。次のコードの一部では、`rpc_call()` はデフォルトで `authnone` (空の認証トークン) を使用するので、認証情報を使用する際に `rpc_call()` では十分でないという点に注意してください。

- `aup_time` - 認証情報のタイム・スタンプ
- `aup_machname` - リモート・クライアントのホスト名
- `aup_uid` - クライアントのリモート・ユーザーの UID
- `aup_gid` - リモート・ユーザーの 1 次 GID
- `aup_gids` - リモート・ユーザーの 2 次グループの配列

認証情報を設定して、それをクライアント・ハンドルの一部として含めるのはクライアント側の責任です。その後で行われる `clnt_call()` へのすべてのコールは、認証情報に従って渡されます。認証されていないクライアントを報告するのはサーバーの役目です。RPC は、情報を伝達するための簡単な方法しか提供しません。クライアントが送信するデータは、認証されてはいますが、暗号化されてはなりません。サービスからの応答も暗号化されていません。認証では、リモート・ホスト名とユーザー識別コードを検証するという簡単な方法しか提供していません。したがって、これを安全でプライベートな通信方法であると見なすことはできません。

```
#include <sys/types.h> /* needed for gid_t and uid_t */
#include <stdlib.h> /* misc. system auth APIs */
#include <unistd.h> /* misc. system auth APIs */
#include <errno.h>

#ifdef NGROUPS_MAX
#define NGROUPS_MAX 16
#endif

char hostname[256]; /* hostname for credentials */
int rslt; /* return value of gethostname() */
gid_t groups[NGROUPS_MAX]; /* array of groups set by getgroups() */
gid_t *aup_gids; /* pointer to array of gid_t */
uid_t uid; /* uid, return value for geteuid() */
gid_t gid; /* gid, return value for getegid() */
int num_groups; /* return value for getgroups(), number of groups set
*/

aup_gids = groups; /* point to the array of groups */
uid = geteuid(); /* get the effective uid of the user */
gid = getegid(); /* get the effect primary gid of the user */

/* get a list of other groups the user is a member of */
/* (int)getgroups(maxgropus, array) */
num_groups = getgroups(NGROUPS_MAX, groups);

/* check return value of getgroups() for error */
if (num_groups == -1) {
    /* print error message and exit */
    fprintf(stderr, "getgroups() failed for %d\n", uid);
    fprintf(stderr, "errno: %d\n", errno);
    return 1;
}

/* (int)gethostname(buffer, buflen) */
rslt = gethostname(hostname, 256);

/* check return value of gethostname() for error */
if (rslt == -1) {
```

```

    /* print error message and exit */
    fprintf(stderr, "gethostname() failed\n");
    fprintf(stderr, "errno: %d\n", errno);
    return 1;
}

/* insert just before clnt_call() */
/* (AUTH *)authsys_create(hostname, uid, gid, num_groups, gid[]); */
clnt->cl_auth = authsys_create(hostname, uid, gid, num_groups, aup_gids);

if (clnt->cl_auth == NULL) {
    /* print error messages and exit */
    fprintf(stderr, "authsys_create() failed\n");
    fprintf(stderr, "errno: %d\n", errno);
    /* clean up */
    clnt_destroy(clnt);
    return 1;
}

```

TI-RPC に関するその他の情報

TI-RPC を使用して分散アプリケーションを設計、インプリメント、および保守するための詳細は、Sun Microsystems 社の ONC+ Developer's Guide  (1997) を参照してください。

rpcgen コンパイラーの使用法の詳細は、Sun Microsystems 社の rpcgen Programmer's Guide  (1997) を参照してください。

TI-RPC サービスのアプリケーション・プログラミング・インターフェース (API) についての詳細は、System API Reference を参照してください。

コーディング例について

この Web ページには、例示のための単純な例として IBM® が提供している小さなプログラムがいくつかあります。このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。このページにあるプログラムはすべて、現状のまま提示されているにすぎません。商業的な使用可能性および特定の目的に対する適合性については、暗黙の保証を含め、いかなる保証も行いません。

付録. 特記事項

本書は米国 IBM が提供する製品およびサービスについて作成したものです。

本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒106-0032
東京都港区六本木 3-2-31
IBM World Trade Asia Corporation
Licensing

以下の保証は、国または地域の法律に沿わない場合は、適用されません。IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態を提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

IBM Corporation
Software Interoperability Coordinator, Department 49XA
3605 Highway 52 N
Rochester, MN 55901
U.S.A.

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、IBM より提供されます。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、またはその他の要求については確認できません。IBM 以外の製品の性能に関する質問は、それらの製品の供給者をお願いします。

IBM の将来の方向または意向に関する記述については、予告なしに変更または撤回される場合があります、単に目標を示しているものです。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。これらの名称はすべて架空のものであり、名称や住所が類似する企業が実在しているとしても、それは偶然にすぎません。

著作権使用許諾:

本書には、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。お客様は、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。お客様は、IBM のアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。

商標

以下は、IBM Corporation の商標です。

AS/400
IBM
Java
Language Environment
Operating System/400
OS/400

Lotus[®]、Freelance、および WordPro は、IBM Corporation の商標です。

ActionMedia[™]、LANDesk[™]、MMX[™]、Pentium[®]、および ProShare[™] は、Intel Corporation の米国およびその他の国における商標または登録商標です。

Microsoft[®]、Windows[®]、Windows NT[®]、および Windows ロゴは、Microsoft Corporation の米国およびその他の国における商標です。

Java[™] およびすべての Java 関連の商標は、Sun Microsystems, Inc の米国およびその他の国における商標です。

UNIX[®] は、The Open Group がライセンスしている米国およびその他の国における登録商標です。

他の会社名、製品名およびサービス名などはそれぞれ各社の商標または登録商標です。

資料に関するご使用条件

お客様がダウンロードされる資料につきましては、以下の条件にお客様が同意されることを条件にその使用が認められます。

個人使用: これらの資料は、すべての著作権表示その他の所有権表示をしていただくことを条件に、非商業的な個人による使用目的に限り複製することができます。ただし、IBM の明示的な承諾をえずに、これらの資料またはその一部について、二次的著作物を作成したり、配布（頒布、送信を含む）または表示（上映を含む）することはできません。

商業的使用: これらの資料は、すべての著作権表示その他の所有権表示をしていただくことを条件に、お客様の企業内に限り、複製、配布、および表示することができます。ただし、IBM の明示的な承諾をえずに、これらの資料の二次的著作物を作成したり、お客様の企業外で資料またはその一部を複製、配布、または表示することはできません。

ここで明示的に許可されているもの以外に、資料や資料内に含まれる情報、データ、ソフトウェア、またはその他の知的所有権に対するいかなる許可、ライセンス、または権利を明示的にも黙示的にも付与するものではありません。

資料の使用が IBM の利益を損なうと判断された場合や、上記の条件が適切に守られていないと判断された場合、IBM はいつでも自らの判断により、ここで与えた許可を撤回できるものとさせていただきます。

お客様がこの情報をダウンロード、輸出、または再輸出する際には、米国のすべての輸出入関連法規を含む、すべての関連法規を遵守するものとします。IBM は、これらの資料の内容についていかなる保証もしません。これらの資料は、特定物として現存するままの状態を提供され、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任なしで提供されます。

これらの資料の著作権はすべて、IBM Corporation に帰属しています。

お客様が、このサイトから資料をダウンロードまたは印刷することにより、これらの条件に同意されたものとさせていただきます。

コードの特記事項情報

本書には、プログラミングの例が含まれています。

IBM は、お客様に、すべてのプログラム・コードのサンプルを使用することができる非独占的な著作使用権を許諾します。お客様は、このサンプル・コードから、お客様独自の特別のニーズに合わせた類似のプログラムを作成することができます。

すべてのサンプル・コードは、例として示す目的でのみ IBM により提供されます。このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。

ここに含まれるすべてのプログラムは、現存するままの状態を提供され、いかなる保証も適用されません。商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任の保証の適用も一切ありません。



Printed in Japan