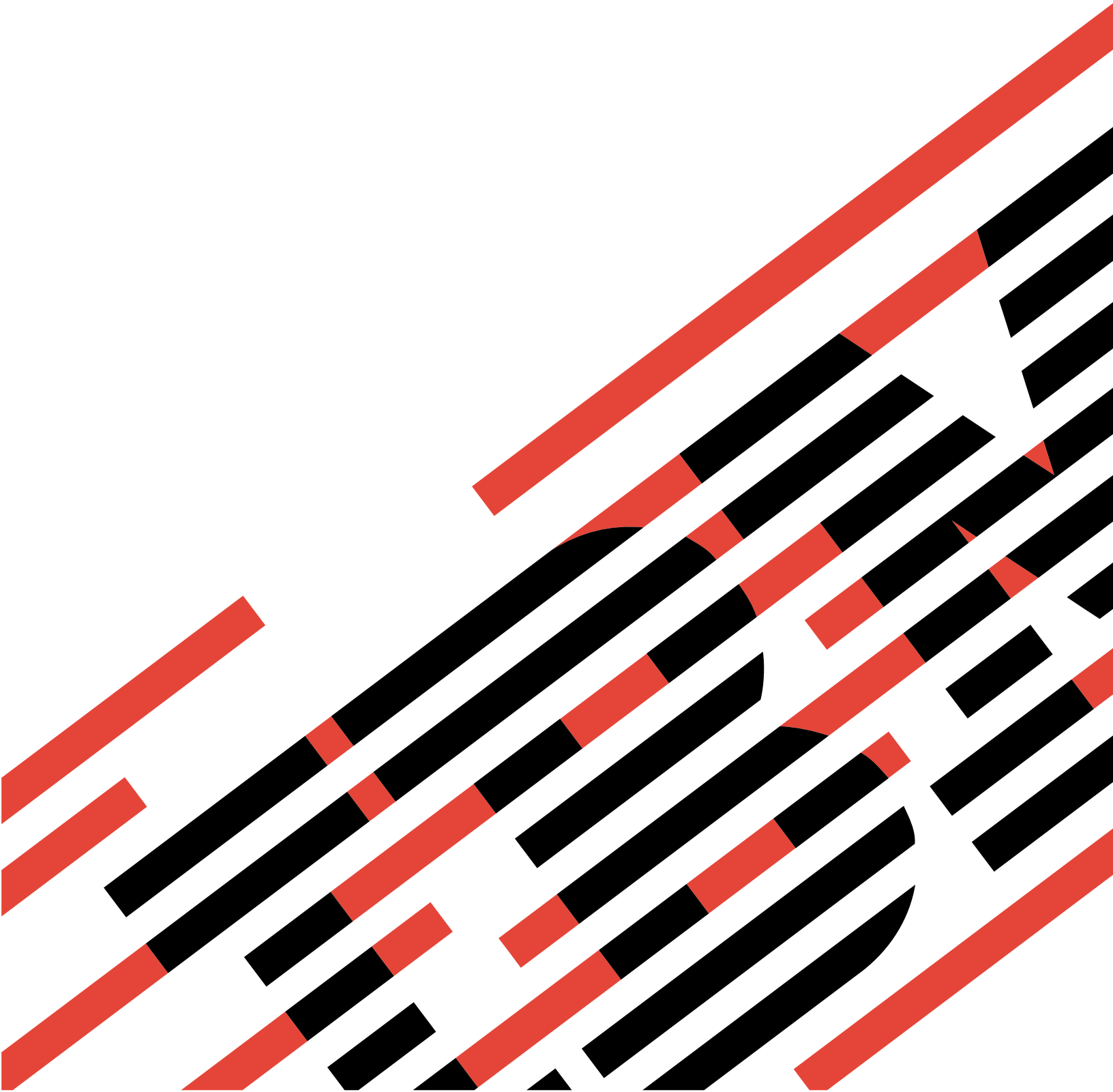
IBM

eserver

iSeries

Sun TI-RPC 분산 어플리케이션



IBM

@server

iSeries

Sun TI-RPC 분산 어플리케이션

목차

제 1 부 Sun TI-RPC를 사용한 분산 어플리케이션 개발

리모트 프로시듀어 호출(RPC)에서는 분산 어플리케이션들이 서로 통신할 수 있는 상위 레벨의 패러다임을 제공합니다. Sun Microsystems에서는 서버 메카니즘에서 클라이언트 어플리케이션을 쉽게 분리시키고 분산시킬 수 있는 개방형 네트워킹 컴퓨터(ONC)를 개발했습니다. 전송 독립 리모트 프로시듀어 호출(TI-RPC) 또는 ONC+ RPC는 가장 최근에 출시할 RPC 버전입니다. TI-RPC는 네트워크층에 사용되는 기초 프로토콜을 추출하는 방법을 제공하므로, 하나의 프로토콜에서 다른 프로토콜로 무리 없는 전이가 이루어지도록 해 줍니다.

AS/400에서 TI-RPC를 사용하여 분산 어플리케이션을 개발하려면 다음 주제를 참조하십시오.

- 5 페이지의 제 2 장『rpcbind 디먼 사용』
- 7 페이지의 제 3 장『rpcgen 컴파일러 사용』
- 9 페이지의 제 4 장『네트워크 선택 메카니즘 사용』
- 11 페이지의 제 5 장『자료 변환 지원 사용』

TI-RPC를 사용하여 분산 어플리케이션을 설계, 구현 및 유지보수하는 것에 관한 자세한 정보는 Sun Microsystems, Inc.(1997)에서 제공하는 ONC+ 개발자 안내서  를 참조하십시오.

코딩 예

서비스 및 클라이언트 어플리케이션 프로그래밍 인터페이스에 대한 예는 다음 주제를 참조하십시오.

- 13 페이지의 제 6 장『예: TI-RPC 코드에 기초한 서비스 어플리케이션 개발』
- 29 페이지의 제 7 장『TI-RPC 코드 예에 기초한 클라이언트 어플리케이션 개발』

제 1 장 이 주제 인쇄

이 문서를 열람하거나 인쇄하기 위해 PDF 버전을 열람하거나 다운로드할 수 있습니다. PDF 파일을 열람하려면 Adobe® Acrobat® Reader가 설치되어 있어야 합니다.

http://www.adobe.com/prodindex/acrobat/readstep.html  에서 사본을 다운로드 받을 수 있습니다.

PDF 버전을 열람하거나 다운로드하려면 Sun TI-RPC 분산 어플리케이션 프로그래밍(약 279KB 또는 58 페이지)을 선택하십시오.

열람이나 인쇄를 위해 PDF를 워크스테이션에 저장하려면 다음과 같이 하십시오.

1. 브라우저에서 PDF를 여십시오(위의 링크를 클릭하십시오).
2. 브라우저 메뉴에서 **파일**을 클릭하십시오.
3. **다른 이름으로 저장...**을 클릭하십시오.
4. PDF를 저장할 디렉토리를 찾으십시오.
5. **저장**을 클릭하십시오.

제 2 장 rpcbind 디먼 사용

클라이언트가 리모트 프로시듀어 호출(RPC) 서비스에 연결하려 할 때, 먼저 RPCBIND 디먼에 서비스 주소를 요청합니다. 이와 같은 방식에서는 주소가 동적으로 처리되기 때문에 클라이언트는 서비스가 어떤 포트를 기다리고있는지 알 필요가 없습니다. 서비스를 사용할 수 있으려면, 이 서비스들을 먼저 rpcbind 디먼에 등록시켜야 합니다. rpcbind 디먼이 비활동 상태이면 서비스를 시작할 수 없고, 클라이언트가 이 서비스를 찾을 수 없습니다.

AS/400에서 rpcbind 디먼을 사용하려면, 다음 타스크를 완료하십시오.

1. OS/400에서 rpcbind 디먼이 실행되고 있는지 확인하십시오.
2. 아직 실행을 시작하지 않았으면 OS/400에서 rpcbind 디먼을 시작하십시오.

OS/400에서 rpcbind 디먼 실행 여부 확인

전송 독립 리모트 프로시듀어 호출(TI-RPC)을 사용하려면, AS/400에서 rpcbind 디먼 작업(QNFSRPCD)이 실행 중인지 확인해야 합니다.

rpcbind 디먼이 실행 중인지 확인하려면 다음 단계를 완료하십시오.

1. OS/400 명령행에서 WRKACTJOB를 입력하십시오.
2. 다음 작업이 있는지 QSYSWRK 서브시스템을 살펴 보십시오.

```
QNFSRPCD The rpcbind daemon
```

rpcbind 디먼을 시작하기 위한 지침은 『OS/400에서 rpcbind 디먼 시작 및 종료』를 참조하십시오.

OS/400에서 rpcbind 디먼 시작 및 종료

rpcbind 디먼(RPCBIND) 명령은 rpcbind 디먼 작업(QNFSRPCD)을 시작합니다.

rpcbind 디먼 작업 시작:

rpcbind 디먼 작업을 시작하려면, 다음 명령을 입력하십시오.

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

주: 이 명령의 선택적 매개변수인 RTVRPCREG는 rpcbind 디먼을 시작할 때 이전에 기록되어 있던 등록 정보를 검색할 것인지를 지정하는 매개변수입니다. 이 매개변수의 디폴트 값은 *NO입니다. rpcbind 디먼을 시작할 때 등록 정보를 검색하려면 *YES를 선택하십시오. 이 명령의 매개변수와 값에 대한 자세한 정보는 온라인 도움말을 참조하십시오.

rpcbind 디먼 작업 종료:

rpcbind 디먼 작업(QNFSRPCD)을 종료하려면, 다음 명령을 입력하십시오.

```
ENDRPCBIND
```

제 3 장 rpcgen 컴파일러 사용

RPCGEN 명령은 리모트 프로시듀어 호출 언어(RPCL)로 작성된 입력 파일로부터 C 코드를 생성합니다. 생성된 C 코드를 사용하여 RPC 프로토콜을 구현할 수 있습니다.

OS/400에서 rpcgen 컴파일러를 사용하려면, 다음 타스크를 완료하십시오.

1. RPCL로 소스 입력 파일을 작성하십시오.

   rpcgen 컴파일러 사용에 대한 자세한 내용은 Sun Microsystems Inc.(1997)에서 제공하는 rpcgen 프로그래머 안내서  를 참조하십시오.

2. OS/400에서 rpcgen 컴파일러를 실행하려면 다음 명령을 입력하십시오.

   ```
   RPCGEN
   ```

   주: 이 명령의 매개변수와 값에 대한 설명은 온라인 도움말을 참조하십시오.

3. rpcgen 컴파일러의 출력을 컴파일하려면 OS/400에서 C 언어 컴파일러를 사용하십시오.

   주: AS/400에서 통합 언어 환경(ILE) C 컴파일러를 사용하는 경우, 출력 파일들을 소스 멤버로 저장해야 합니다.

제 4 장 네트워크 선택 메카니즘 사용

네트워크 선택 메카니즘은 어플리케이션을 실행시킬 전송(transport)을 선택할 수 있게 해 줍니다. /etc/netconfig 파일은 호스트에서 사용할 수 있는 전송을 유형으로 식별하여 나열한 데이터베이스입니다. 전송은 지정된 순서로 /etc/netconfig 파일에서 사용할 수 있습니다. 네트워크 선택 어플리케이션 프로그래밍 인터페이스(API)에 대한 자세한 정보는 시스템 API 참조서를 참조하십시오.

iSeries Navigator에서 OS/400의 /etc/netconfig 파일에 액세스하려면, 다음 단계를 완료하십시오.

1. PC에 설치되어 있는 iSeries Navigator를 여십시오.
2. 네트워크 폴더를 확장하십시오.
3. 서버 폴더를 확장하십시오.
4. OS/400 서버를 클릭하십시오.
5. 오른쪽 마우스 버튼으로 RPC를 클릭하여 표시된 팝업 메뉴에서 등록 정보를 선택하십시오.
6. RPC 전송 탭을 클릭하십시오.

주: 이 정보를 열람하려면 *IOSYSCFG 권한이 있어야 합니다.

제 5 장 자료 변환 지원 사용

전송 독립 리모트 프로시듀어 호출(TI-RPC) 어플리케이션 프로그래밍 인터페이스(API)는 모두 OS/400에서 자국어 지원(NLS)에 사용 가능합니다. 이 지원은 eXternal Data Representation(XDR) 기능 리스트에 추가되었습니다. 이 기능들을 사용하여 서로 다른 코드 페이지의 클라이언트와 서비스간의 자료 통신이 가능합니다. 시스템 관리자는 /etc/rpcnls 파일을 유지보수하여 코드 페이지와 리모트 클라이언트를 연관시킵니다. XDR 기능은 /etc/rpcnls 파일의 정보를 사용하여 내재적 자료 변환을 제공합니다. 다음의 XDR 기능에는 내장된 내재적 자료 변환 루틴이 들어 있습니다.

- `xdr_char()` (1바이트의 경우에만 해당)
- `xdr_u_char()` (1바이트의 경우에만 해당)
- `xdr_double_char` (1바이트 및 2바이트의 경우에 해당)
- `xdr_string()` (1바이트 및 2바이트의 경우에 해당)
- `xdr_wrapstring()` (1바이트 및 2바이트의 경우에 해당)

각각의 전송 독립 리모트 프로시듀어 호출(TI-RPC) 어플리케이션 프로그래밍 인터페이스(API)에서의 자료 변환 지원에 대한 자세한 정보는 시스템 API 참조서를 참조하십시오.

iSeries Navigator에서 OS/400의 /etc/rpcnls 파일에 액세스하려면, 다음 단계를 완료하십시오.

1. PC에 설치되어 있는 iSeries Navigator를 여십시오.
2. 네트워크 폴더를 확장하십시오.
3. 서버 폴더를 확장하십시오.
4. OS/400 서버를 클릭하십시오.
5. 오른쪽 마우스 단추로 RPC를 클릭하여 표시된 팝업 메뉴에서 등록 정보를 선택하십시오.
6. 자료 변환 지원 탭을 클릭하십시오.

주: 이 정보를 열람하려면 *IOSYSCFG 권한이 있어야 합니다.

제 6 장 예: TI-RPC 코드에 기초한 서비스 어플리케이션 개발

전송 독립 리모트 프로시듀어 호출(TI-RPC) 프로그래밍은 OS/400에서 분산 클라이언트-서버에 근거한 어플리케이션을 개발하는 데 효율적인 방법을 제공합니다.

OS/400에서 서비스 어플리케이션을 개발하려면, 다음 코드 예를 지침으로 사용하십시오.

- 『예: TI-RPC 기초 레벨 서비스 API』
- 16 페이지의 『예: TI-RPC 최상위 레벨 서비스 API』
- 23 페이지의 『예: TI-RPC 중간 레벨 서비스 API』
- 25 페이지의 『예: TI-RPC 전문가 레벨 서비스 API』
- 26 페이지의 『예: TI-RPC 서비스에 인증 추가』

관련 정보는 다음과 같습니다.

- 1 페이지의 제 1 부 『Sun TI-RPC를 사용한 분산 어플리케이션 개발』
- 29 페이지의 제 7 장 『TI-RPC 코드 예에 기초한 클라이언트 어플리케이션 개발』

예: TI-RPC 기초 레벨 서비스 API

다음의 코드 예에서는 전송 독립 리모트 프로시듀어 호출(TI-RPC) 서비스를 개발하는 데 사용되는 기초 레벨 서비스 어플리케이션 프로그래밍 인터페이스(API) 중 하나를 보여줍니다.

이 코드 예에서 프로시듀어가 각기 독립적으로 등록되는 방식에 주의하십시오. 기초 레벨에서 서비스는 복수의 프로시듀어를 가질 수 있지만, 각각을 별도로 등록해야 합니다. 서비스를 등록하지 않으면 프로시듀어도 모두 등록되지 않습니다. 나머지 프로시듀어를 제외하고 하나의 프로시듀어만 개별적으로 등록을 취소할 수 있는 방법은 없습니다.

이 레벨은 프로시듀어의 수가 적은 서비스에 적합합니다. 또한 최종 프로시듀어 중 일부 제한된 수의 프로시듀어만을 사용하여 더 큰 서비스를 프로토타입화할 때 유용합니다. 다른 모든 서비스 레벨에서와 같이, 서비스에서 마지막 호출은 svc_run()이어야 하고, 이것으로 선택 대기(클라이언트로부터의 연결을 기다림) 상태가 됩니다.

```
#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() */
```

예: TI-RPC 최상위 레벨 서비스 API

다음의 코드 예에서는 전송 독립 리모트 프로시듀어 호출(TI-RPC) 서비스를 개발하는 데 사용되는 최상위 레벨 서비스 어플리케이션 프로그래밍 인터페이스(API)를 보여줍니다.

최상위 레벨에서의 서비스 개발은 개발자가 디스패치 루틴을 작성해야 하기 때문에 더 복잡합니다. 이 레벨에서 서비스 요구가 들어오면 디스패치 루틴이 호출됩니다. 디스패치 루틴은 인수들을 수집하고, 올바른 로컬 프로시듀어를 호출하며, 오류와 결과를 모두 구하여 그 정보를 클라이언트로 보내주어야 합니다. 일단 디스패치 함수가 작성되면, 쉽게 복사할 수 있고 약간의 수정만으로 다른 서비스에서 사용할 수 있습니다.

수정하지 않고도 최상위, 중간 및 전문가 계층에서 같은 디스패치 함수를 사용할 수 있습니다. 다음 예에서는 디스패치 함수가 다른 로컬 함수와 함께 이 파일에 들어 있습니다. 서비스를 실행하기 전에 두 파일 모두 컴파일하고 링크시켜야 합니다. 다른 계층 위에 있는 최상위 레벨의 장점은 네트워크 선택 API를 사용하는 대신 nettype을 스트링으로 지정할 수 있는 기능이 있다는 점입니다. 최상위 레벨 API이 호출되고나면 서비스가 작성되고 디스패치 함수에 바인드된 후 rpcbind 서비스에 등록됩니다.

```
#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\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() */

/* 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() */
```

예: TI-RPC 중간 레벨 서비스 API

다음의 코드 예에서는 전송 독립 리모트 프로시듀어 호출(TI-RPC) 서비스를 개발하는 데 사용되는 중간 레벨 서비스 어플리케이션 프로그래밍 인터페이스(API)를 보여줍니다.

디스패치 함수는 변경하지 않고도 사용할 수 있습니다. 중간 레벨과 최상위 레벨 사이의 유일한 차이는 네트워크 선택 API 사용 여부입니다. Sun Microsystems, Inc. (1997)이 레벨 역시 서비스를 작성하고 rpcbind 디먼에 바인드 및 등록합니다.

```
#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\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() */
```

예: TI-RPC 전문가 레벨 서비스 API

다음의 코드 예에서는 전송 독립 리모트 프로시듀어 호출(TI-RPC) 서비스를 개발하는 데 사용되는 전문가 레벨 서비스 어플리케이션 프로그래밍 인터페이스(API)를 보여줍니다.

전문가 레벨에서 프로그래머는 서비스의 송신 및 수신 버퍼 크기를 지정할 수 있습니다. svc_tli_create()를 호출하면 서비스 핸들은 작성되지만, 등록되거나 rpcbind 디먼에 바인드되지는 않습니다. 서비스가 올바로 작동하기 위해서는 프로그래머가 svc_reg()를 호출해야 합니다. 중간 레벨에서와 마찬가지로, 네트워크 선택 API를 사용하여 서비스의 전송 정보를 검색할 수 있는 옵션이 있습니다.

```
#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(%s)\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() */
```

예: TI-RPC 서비스에 인증 추가

다음 코드는 리모트 프로시듀어 호출(RPC)에서 인증 시스템이 작동하는 방식을 보여줍니다. 시스템은 OS/400에서 제공하는 유일한 인증 방법입니다. 다음 정보를 설정하여 클라이언트에서 clnt_call()을 사용하는 서비스로 전달하십시오. 다음 코드에 있어서 인증 정보를 사용할 때 rpc_call()이 충분하지 않다는 점에 주의해야 하는데, 이것은 디폴트로 authnone(빈 인증 토큰)을 사용하기 때문입니다.

- aup_time - 인증 정보 시간소인
- aup_machname - 리모트 클라이언트의 호스트명
- aup_uid - 클라이언트의 리모트 사용자의 UID
- aup_gid - 리모트 사용자의 1차 GID
- aup_gids - 리모트 사용자의 2차 그룹 배열

인증 정보는 원격 요구의 일부로 직접 서비스로 들어옵니다. 이 정보를 분석하여 클라이언트가 신뢰할 수 있는 시스템과 신뢰할 수 있는 사용자에게서 나온 것인지를 검증하는 것은 서버가 해야 할 일입니다. 인증 유형이 잘못되었거나 서버가 받아들이기에는 신뢰가 부족한 경우, svcerr_weakauth()로 오류를 송신하여 클라이언트에 알립니다.

```
#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 ((strcmpi("remote1", remote_machine) != 0) &&
    (strcmpi("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 trst */
      /* send back an authentication error the client */
      svcerr_weakauth(svc);
      return;

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

} /* end of else (tusted hostname) */

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

제 7 장 TI-RPC 코드 예에 기초한 클라이언트 어플리케이션 개발

전송 독립 리모트 프로시듀어 호출(TI-RPC) 프로그래밍은 OS/400에서 분산 클라이언트-서버에 근거한 어플리케이션을 개발하는 데 효율적인 방법을 제공합니다. TI-RPC 서비스 어플리케이션 프로그래밍 인터페이스(API)에 대한 자세한 정보는 시스템 API 참조서를 참조하십시오.

OS/400에서 클라이언트 어플리케이션을 개발하려면, 다음 코드 예를 지침으로 사용하십시오.

- 『예: TI-RPC 기초 레벨 클라이언트 API』
- 35 페이지의 『예: TI-RPC 최상위 레벨 클라이언트 API』
- 39 페이지의 『예: TI-RPC 중간 레벨 클라이언트 API』
- 44 페이지의 『예: TI-RPC 전문가 레벨 클라이언트 API』
- 50 페이지의 『예: TI-RPC 클라이언트에 인증 추가』

관련 정보는 다음과 같습니다.

- 1 페이지의 제 1 부 『Sun TI-RPC를 사용한 분산 어플리케이션 개발』
- 13 페이지의 제 6 장 『예: TI-RPC 코드에 기초한 서비스 어플리케이션 개발』

예: TI-RPC 기초 레벨 클라이언트 API

다음의 코드 예에서는 전송 독립 리모트 프로시듀어 호출(TI-RPC) 어플리케이션을 개발하는 데 사용되는 전문가 레벨 클라이언트 어플리케이션 프로그래밍 인터페이스(API)를 보여줍니다.

기초 레벨 클라이언트 API는 클라이언트의 작성, 제어, 사용, 제거를 모두 한 호출에서 처리하므로 가장 빠르고 짧은 코드 세트입니다. 이것은 편리하기는 하지만 클라이언트 핸들로 처리할 수 있는 사용자 정의가 불가능합니다. 시간종료 및 버퍼 크기에는 디폴트 값이 사용되는데, 이것이 기초 레벨과 기타 다른 레벨과의 가장 큰 차이점입니다.

```
#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, "clnt_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, "clnt_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, "clnt_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("%.2ld - GET_UID             %.2ld - GET_UID_STRING\n",
          GET_UID, GET_UID_STRING);
   printf("%.2ld - GET_SIZE            %.2ld - GET_MTIME\n",
          GET_SIZE, GET_MTIME);
   printf("%.2ld - GET_MTIME_STRING    %.2ld - GET_CODEPAGE\n",
```

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

}
```

예: TI-RPC 최상위 레벨 클라이언트 API

다음의 코드 예에서는 전송 독립 리모트 프로시듀어 호출(TI-RPC) 어플리케이션을 개발하는 데 사용되는 최상위 레벨 클라이언트 어플리케이션 프로그래밍 인터페이스(API)를 보여줍니다.

최상위 레벨에서는 클라이언트 핸들을 먼저 작성하고 나서 이것을 사용하거나 수정할 수 있습니다. 최상위 레벨 API는 사용이 간편하며, 기초 레벨에 비해 더 많은 조작 및 오류 처리가 가능합니다.

```
#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("%.2ld - GET_UID             %.2ld - GET_UID_STRING\n",
          GET_UID, GET_UID_STRING);
   printf("%.2ld - GET_SIZE            %.2ld - GET_MTIME\n",
          GET_SIZE, GET_MTIME);
   printf("%.2ld - GET_MTIME_STRING    %.2ld - GET_CODEPAGE\n",
          GET_MTIME_STRING, GET_CODEPAGE);
   printf("%.2ld - GET_OBJTYPE         %.2ld - GET_FILETYPE\n",
          GET_OBJTYPE, GET_FILETYPE);
   printf("%.2ld - END_SERVER          %.2d - EXIT\n",
          END_SERVER, EXIT);

}
```

예: TI-RPC 중간 레벨 클라이언트 API

다음의 코드 예에서는 전송 독립 리모트 프로시듀어 호출(TI-RPC) 어플리케이션을 개발하는 데 사용되는 중간 레벨 클라이언트 어플리케이션 프로그래밍 인터페이스(API)를 보여줍니다.

클라이언트의 중간 레벨은 서비스에서와 마찬가지로 같은 경로를 따릅니다. 예를 들어, 간단한 텍스트 스트링을 전달하는 대신 네트워크 선택 API를 사용하여 전송 정보를 구하는 것은 프로그래머의 책임입니다.

```
#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("%.2ld - GET_UID             %.2ld - GET_UID_STRING\n",
          GET_UID, GET_UID_STRING);
   printf("%.2ld - GET_SIZE            %.2ld - GET_MTIME\n",
          GET_SIZE, GET_MTIME);
   printf("%.2ld - GET_MTIME_STRING    %.2ld - GET_CODEPAGE\n",
          GET_MTIME_STRING, GET_CODEPAGE);
   printf("%.2ld - GET_OBJTYPE         %.2ld - GET_FILETYPE\n",
          GET_OBJTYPE, GET_FILETYPE);
   printf("%.2ld - END_SERVER          %.2d - EXIT\n",
          END_SERVER, EXIT);

}
```

예: TI-RPC 전문가 레벨 클라이언트 API

다음의 코드 예에서는 전송 독립 리모트 프로시듀어 호출(TI-RPC) 어플리케이션을 개발하는 데 사용되는 전문가 레벨 클라이언트 어플리케이션 프로그래밍 인터페이스(API)를 보여줍니다.

클라이언트 API 개발에서 전문가 레벨이 가장 복잡합니다. 사용자 정의도 가장 많이 할 수 있습니다. 또한 버퍼 *크기*를 클라이언트 API에 맞게 조정할 수 있는 유일한 레벨이기도 합니다. 이 레벨에서는 이름-주소 변환

API를 사용하거나 다른 전문가 레벨 API 중 하나를 사용하여 프로그래머가 연결 클라이언트의 범용 주소를 설정해야 합니다. 어느 쪽이든, 이 레벨에서는 좀 더 많은 작업을 수행해야 하지만, 프로그래머가 클라이언트 어플리케이션을 실행 환경에 맞게 수정할 수 있습니다.

```
#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("%.2ld - GET_UID             %.2ld - GET_UID_STRING\n",
          GET_UID, GET_UID_STRING);
   printf("%.2ld - GET_SIZE            %.2ld - GET_MTIME\n",
          GET_SIZE, GET_MTIME);
   printf("%.2ld - GET_MTIME_STRING    %.2ld - GET_CODEPAGE\n",
          GET_MTIME_STRING, GET_CODEPAGE);
   printf("%.2ld - GET_OBJTYPE         %.2ld - GET_FILETYPE\n",
          GET_OBJTYPE, GET_FILETYPE);
   printf("%.2ld - END_SERVER          %.2d - EXIT\n",
          END_SERVER, EXIT);

}
```

예: TI-RPC 클라이언트에 인증 추가

다음 코드는 리모트 프로시듀어 호출(RPC)에서 인증 시스템이 작동하는 방식을 보여줍니다. 시스템은 OS/400에서 제공하는 유일한 인증 방법입니다. 다음 정보를 설정하여 클라이언트에서 clnt_call()을 사용하는 서비스로 전달하십시오. 다음 코드에 있어서 인증 정보를 사용할 때 rpc_call()이 충분하지 않다는 점에 주의해야 하는데, 이것은 디폴트로 authnone(빈 인증 토큰)을 사용하기 때문입니다.

- aup_time - 인증 정보 시간소인
- aup_machname - 리모트 클라이언트의 호스트명
- aup_uid - 클라이언트의 리모트 사용자의 UID
- aup_gid - 리모트 사용자의 1차 GID
- aup_gids - 리모트 사용자의 2차 그룹 배열

인증 정보를 설정하고 이것을 클라이언트 핸들의 일부로 만드는 것은 클라이언트가 해야 하는 일입니다. 그리고 나서, clnt_call()에 대한 후속 호출이 발생할 때 인증 정보도 함께 전달됩니다. 권한이 없는 클라이언트를 보고하는 것은 서버가 할 일입니다. RPC는 단지 정보를 통신하는 간단한 방법을 제공합니다. 클라이언트가 송신하는 자료는 인증된 것으로서 암호화된 것이 아닙니다. 서비스로부터의 응답 역시 암호화되지 않습니다. 인증은 리모트 호스트명과 사용자 식별을 검증할 수 있는 간단한 방법을 제공합니다. 그러므로 기밀이 유지되는 안전한 통신 방법이라고 할 수 없습니다.

```
#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>

#ifndef 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;
}
```

제 8 장 TI-RPC에 대한 기타 정보

TI-RPC를 사용하여 분산 어플리케이션을 설계, 구현 및 유지보수하는 것에 관한 자세한 정보는 Sun Microsystems, Inc.(1997)에서 제공하는 ONC+ 개발자 안내서  를 참조하십시오.

rpcgen 컴파일러 사용에 대한 자세한 내용은 Sun Microsystems Inc.(1997)에서 제공하는 rpcgen 프로그래머 안내서  를 참조하십시오.

TI-RPC 서비스 어플리케이션 프로그래밍 인터페이스(API)에 대한 자세한 정보는 시스템 API 참조서를 참조하십시오.

코딩 예에 대한 정보

이 웹 페이지에는 예시를 위해 IBM이 간단한 예로 제공하는 작은 프로그램들이 들어 있습니다. 이 예들은 모든 조건 하에서 완벽하게 테스트된 것들이 아닙니다. 그러므로 IBM은 이 프로그램들의 신뢰성, 서비스 및 기능을 보장할 수 없습니다. 여기에 나오는 프로그램들은 모두 "있는 그대로" 제공됩니다. 상업성 또는 특정 목적에 부합성에 대한 내재적 보장도 하지 않습니다.

Printed in U.S.A