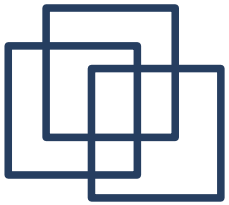
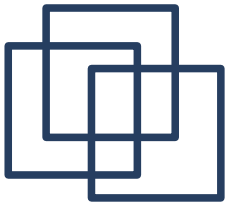


Programowanie z użyciem RPC

1. Zastosowanie modelu zdalnie wywoływanych procedur.
2. Biblioteka RPC.
3. Podział programu na część lokalną i zdalną.
4. Narzędzie rpcgen.
5. Generowanie programu rozproszonego.



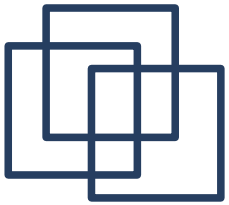
Zastosowanie modelu zdalnych wywołań procedur

Dzięki ogólności modelu RPC programista może go zastosować na różnych etapach projektowania i realizacji aplikacji rozproszonej:

• **technika konstrukcji specyfikacji programu** – dane przesyłanie pomiędzy klientem i serwerem są specyfikowane jako argumenty wywołania oraz wartości zwracane przez procedurę,

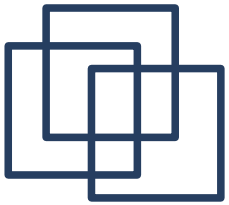
• **projektowanie programu** – poszczególne komunikaty zaprojektowanego protokołu odpowiadają wywołaniom procedur,

• **projekt pojęciowy i implementacja jawnie oparta na modelu RPC** – przesyłanie danych między klientem i serwerem programuje się dokładnie według specyfikacji proceduralnej. Argumenty są kodowane zgodnie ze standardem zewnętrznej reprezentacji danych. Ich typy odpowiadają dokładnie typom danych określonych w specyfikacji,



Zastosowanie modelu zdalnych wywołań procedur

- **projekt i implementacja oprogramowania które, jest konstruowane od podstaw** – programista buduje zwykły program użytkowy rozwiązujący dany problem po czym dzieli go na procedury, które będą działać na różnych komputerach.
- **projekt i implementacja oprogramowania, które wykorzystuje standardowe procedury biblioteczne** – programista pisze program rozproszony zgodnie ze specyfikacją Sun RPC. Zarejestrowanie zdalnych programów, konstruowanie i przesyłanie komunikatów realizujących wywołania procedur jest realizowane przez procedury biblioteczne,
- **automatyczna generacja kodu opartego na modelu RPC** – części kodu zawierające wywołania procedur bibliotecznych realizujące komunikacje są generowane automatycznie.



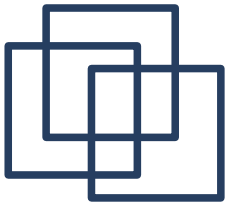
Oprogramowanie pomocnicze

Implementacje mechanizmu Sun RPC zawierają oprogramowanie pomocnicze, pozwalające w znacznym stopniu zmniejszyć nakład pracy programisty. Można je podzielić na następujące grupy:

• procedury biblioteczne XDR umożliwiające konwersję typów prostych (**`int`**, **`float`**) oraz agregatów danych (tablice struktury),

• procedury RPC służące do wywołania i odebrania wyników ze zdalnej procedury

• generator programów (procedur łącznikowych) produkujący część plików źródłowych potrzebnych do zbudowania programu rozproszonego.



Biblioteka procedur RPC

Biblioteka procedur RPC **rpc/rpc.h** zawiera większość funkcji potrzebnych dla programów korzystających z tego mechanizmu.

Przykładowo, aby wywołać zdalną procedurę można użyć **callrpc()**.

```
int callrpc(char *host, u_long prnum, u_long vnum,
        u_long pnum, xdrproc_t inproc, char *in,
        xdrproc_t outproc, char *out);
```

***host** – nazwa lub adres komputera,

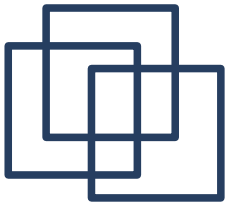
prnum, vnum, pnum – numery programu, wersji i procedury,

inproc, outproc – wskaźniki do procedur konwertujących dane,

in, out – bufory zawierające argumenty oraz miejsce na wynik.

Wartość zwracana: 0 w przypadku sukcesu lub numer błędu, który można odczytać używając funkcji **clnt_perrno()**.

Uwaga: Procedura używa protokołu UDP.



Biblioteka procedur RPC

Program serwera w celu zarejestrowania procedury może wywołać:

```
int registerrpc(u_long prnum, u_long vnum, u_long pnum,
                char *procname, xdrproc_t inproc,
                xdrproc_t outproc);
```

prnum, vnum, pnum – numery programu, wersji i procedury,

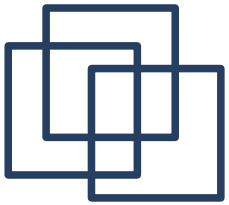
procname – nazwa funkcji realizującej zdalną procedurę,

inproc, outproc – wskaźniki do procedur służących do konwersji danych,

Wartość zwracana: 0 w przypadku pomyślnej rejestracji, -1 w przeciwnym przypadku.

Uwaga: Procedura używa protokołu UDP.

Aby uruchomić serwer wystarczy wywołać funkcję **svc_run()**.



Biblioteka procedur RPC

Inne popularne funkcje to:

```
CLIENT *clnt_create(char *host, u_long prog,
                    u_long vers, char *proto);
```

zwraca „uchwyt” obsługujący połączenie ze zdalnym programem.

```
AUTH *authunix_create(char *host, int uid, int gid,
                      int len, int *gids)
```

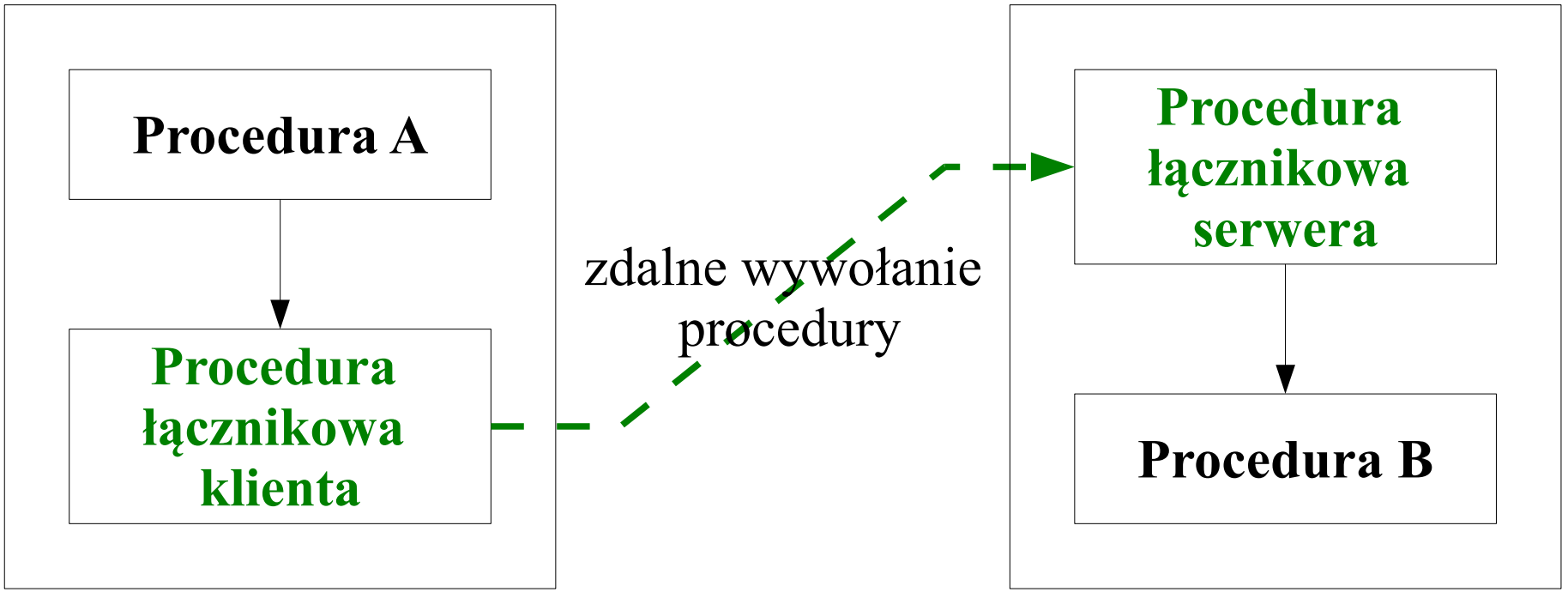
używana do autoryzacji poprzez mechanizm związany z systemem UNIX.

Można konstruować aplikacje, które bezpośrednio wywołują procedury biblioteki RPC, jednak nie jest to najpopularniejsza technika. Zazwyczaj korzysta się z automatycznego generatora. Kod wyprodukowany przez ten generator zawiera wywołania procedur biblioteki RPC.

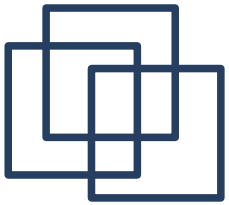
Podział programu na część lokalną i zdalną

Z każdą procedurą jest związany zbiór parametrów formalnych, a z każdym wywołaniem zbiór argumentów. Liczba oraz typy argumentów podanych przez procedurę wywołującą muszą być zgodne z liczbą i typami parametrów formalnych – parametry określają interfejs między procedurą wywoływaną i wywołującą.

Podział programu na część lokalną i zdalną



Procedury łącznikowe dodane do programu realizują zdalne wywołanie procedury. Interfejs tych procedur jest taki sam jak pierwotny interfejs wywołania, dlatego ani procedura wywołująca ani procedura wywoływana nie wymaga modyfikacji.

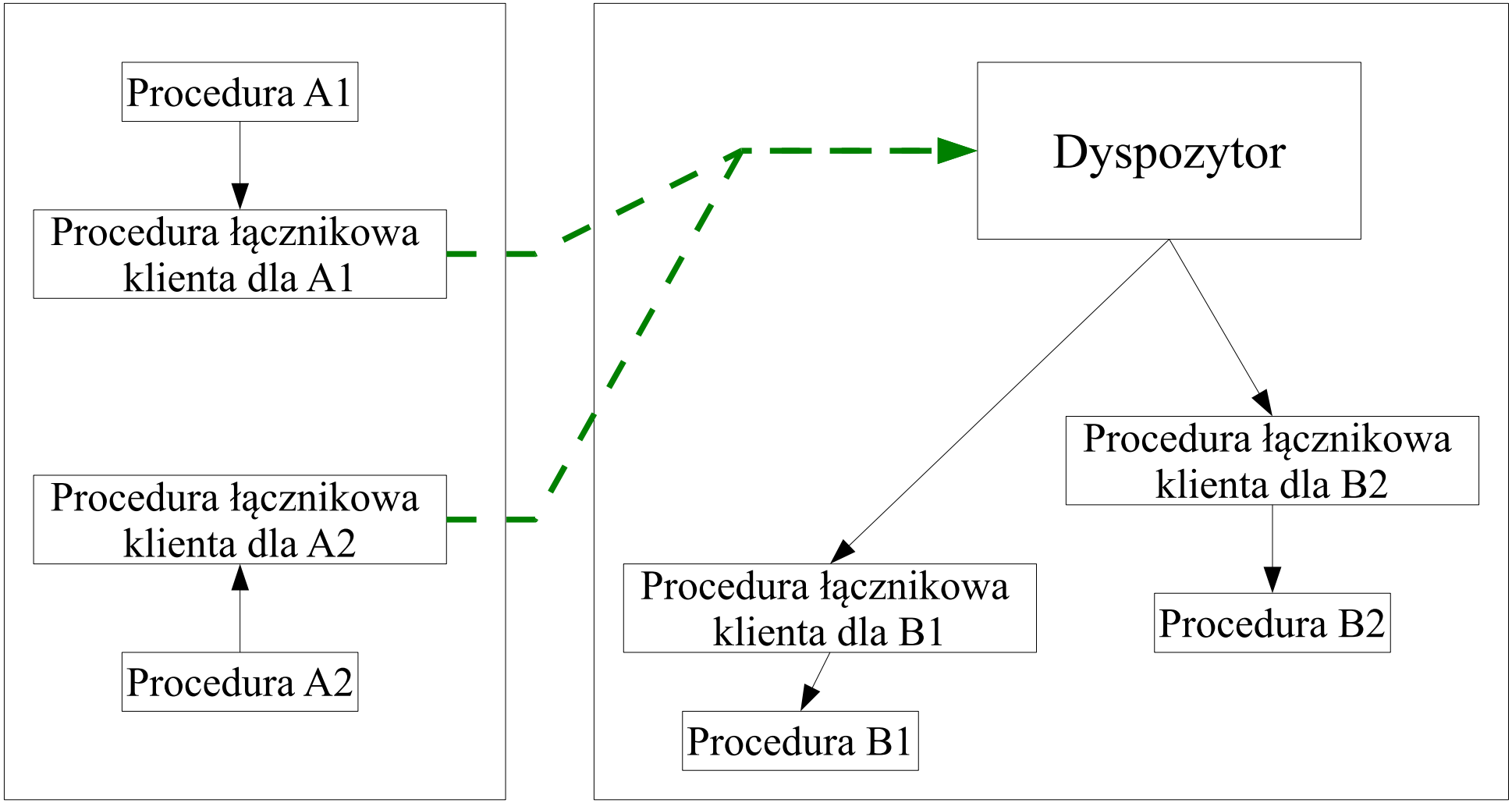


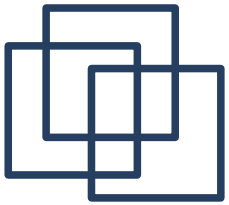
Dodatkowy kod realizujący komunikację RPC

Aby przenieść procedurę na inny komputer do programu należy wstawić dodatkowy kod. Po stronie klienta będzie on odpowiedzialny za serializację i konwersję argumentów, wysłanie komunikatu do komputera odległego a następnie odebranie i konwersję wyników. Po stronie serwera dodatkowy moduł przyjmuje zgłoszenia RPC, przekształca argumenty wywołania i przekazuje je właściwej procedurze. Po jej wykonaniu przekształca i odsyła wyniki.

Te dodatkowe moduły mogą mieć postać procedur, w których będą ukryte wszystkie szczegóły dotyczące komunikacji między procedurami lokalnymi i odległymi. W ten sposób kod wykonujący operacje RPC pozostanie oddzielony od kodu aplikacji.

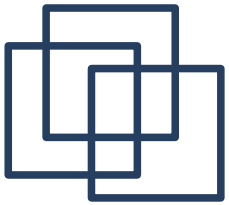
Podział programu na część lokalną i zdalną





Podział programu - podsumowanie

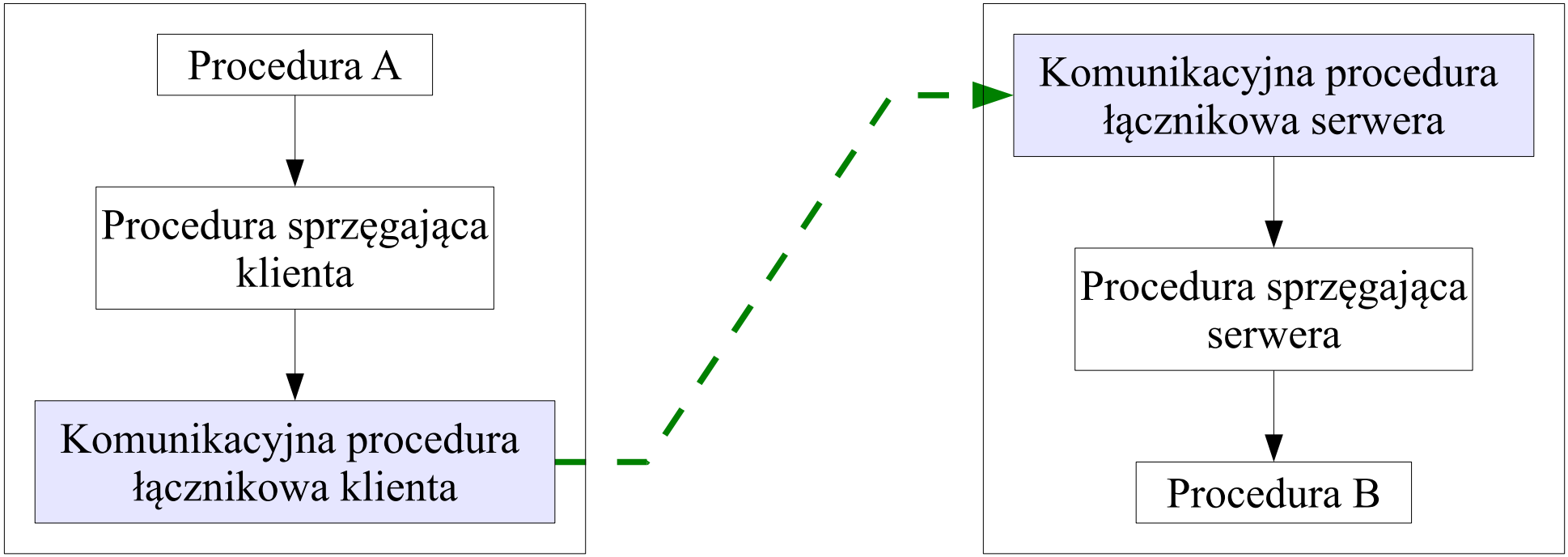
Aby uzyskać rozproszoną wersję aplikacji, trzeba przenieść jedną lub więcej procedur do komputera odległego. Dodanie procedur łącznikowych jest rozwiązaniem pozwalającym pozostawić procedury wywołujące i wywoływane w ich pierwotnej postaci, pod warunkiem, że każda procedura łącznikowa po stronie klienta będzie nazwana tak samo jak procedura wywoływana w pierwotnym programie.



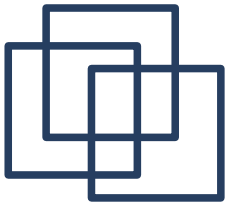
Zastosowanie narzędzia rpcgen

W implementacjach Sun RPC jest dostępne narzędzie pozwalające znacznie zmniejszyć nakład pracy potrzebny do skonstruowania aplikacji rozproszonej – rpcgen. Na wejściu otrzymuje on plik specyfikacji zawierający deklaracje stałych, globalnych typów danych, zmiennych globalnych i zdalnie wywoływanych procedur. Pliki kodu źródłowego otrzymane na wyjściu to przede wszystkim procedury łącznikowe dla strony klienta i serwera zawierający kod realizujący serializację argumentów, wysyłanie i odbieranie komunikatów RPC, konwersję danych pomiędzy reprezentacją natywną i zewnętrzną.

Procedury łącznikowe przy użyciu rpcgen



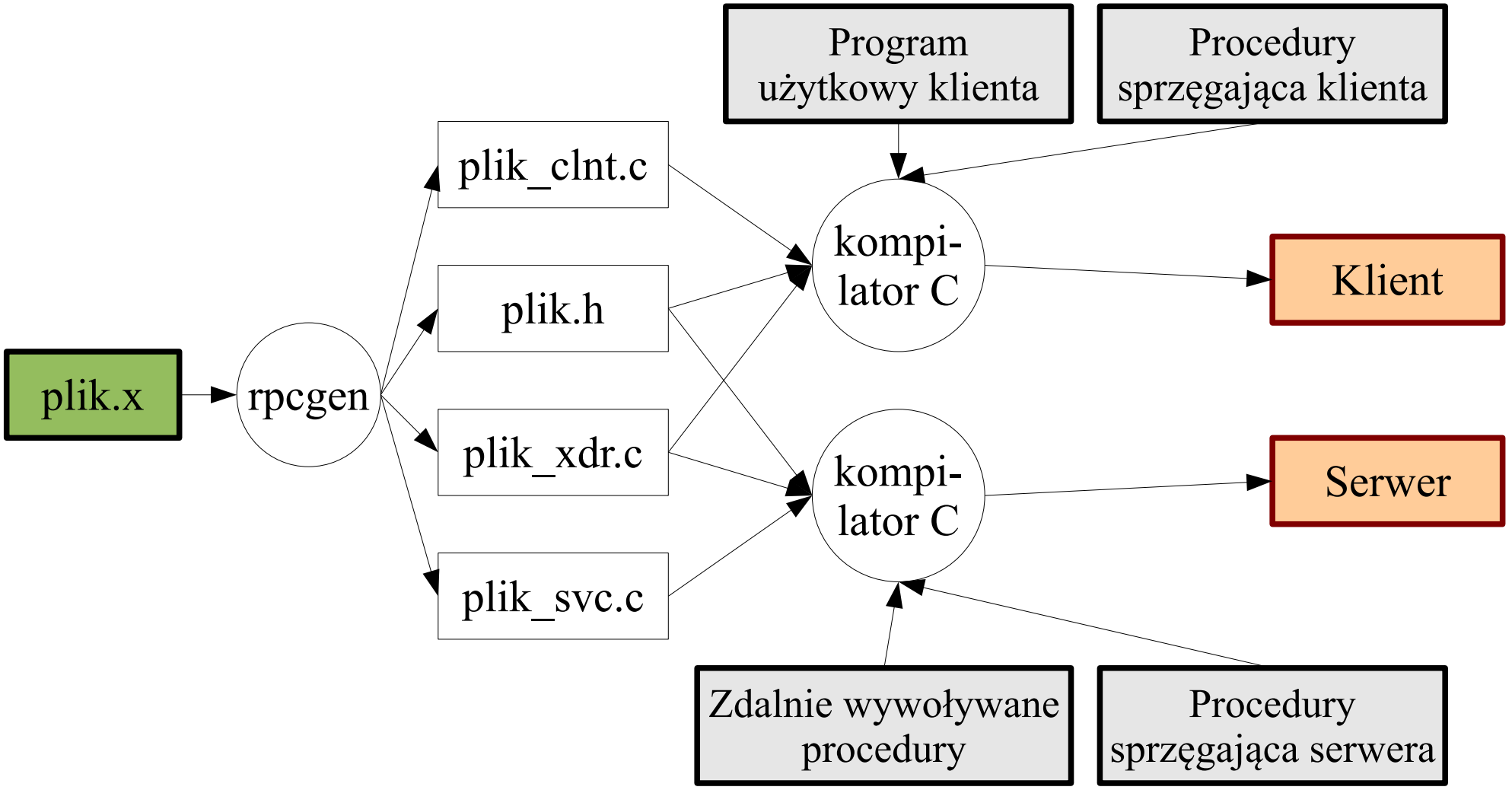
Procedura łącznikowa zostaje podzielona na część komunikacyjną i sprzęgającą. Część komunikacyjna jest niemal identyczna dla wszystkich aplikacji rozproszonych. Część sprzęgająca pełni rolę interfejsu pomiędzy procedurą komunikacyjną a programem.

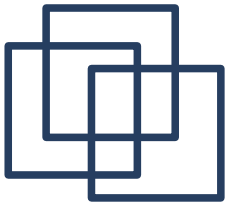


Pliki związane z rpcgen

Nazwa pliku	Zawartość
plik.x	**plik wejściowy – specyfikacja zdalnego programu**
plik.h	deklaracje typów używanych w wygenerowanym kodzie.
plik_xdr.c	wywołania procedur XDR używanych przez klienta i program serwera w celu w celu dokonania serializacji argumentów.
plik_clnt.c	łącznikowa procedura komunikacyjna strony klienta.
plik_svc.c	łącznikowa procedura komunikacyjna strony serwera.

Struktura programu rozproszonego (rpcgen)

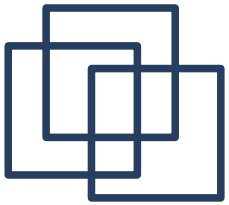




Generowanie programu rozproszonego (przykład)

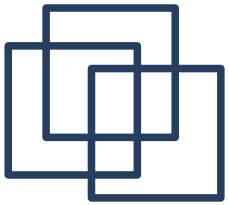
Budowanie aplikacji rozproszonej z wykorzystaniem narzędzia rpcgen odbywa się w ośmiu krokach:

1. Skonstruowanie i przetestowanie zwykłego (działającego lokalnie) programu użytkowego.
2. Podział programu na część lokalną i zdalną.
3. Napisanie specyfikacji zdalnie wywoływanego programu wykorzystywanej przez rpcgen.
4. Użycie generatora rpcgen do wyprodukowania plików źródłowych wykorzystywanych do budowy klienta i serwera.



Generowanie programu rozproszonego (przykład)

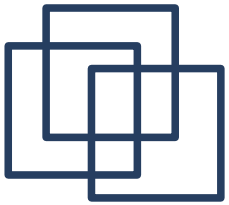
5. Implementacja procedur sprzęgających po stronie klienta i serwera.
6. Uzupełnienie, kompilacja i konsolidacja (linkowanie) plików składających się na program kliencki.
7. Uzupełnienie, kompilacja i konsolidacja (linkowanie) plików składających się na program serwera.
8. Uruchomienie serwera na komputerze odległym i klienta (jednego lub wielu) na komputerach lokalnych.



Krok 1: program lokalny

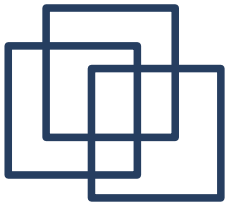
Przykładowy program przedstawia prostą implementację dla zbioru słów.

```
#include<stdio.h>
#define WORDLEN 50

int main(int argc, char **argv){
    char cmdline[WORDLEN+3];
    char cmd;
    int i;

    while(1){
        scanf("%s", cmdline);
        if (strlen(cmdline)<1){
            printf("brak komendy\n");
            continue;
        }
        cmd = cmdline[0];
        switch(cmd){
            case 'I':
                i = init();
                printf("zbior zainicjowany (%d)\n", i);
                break;
```



Krok 1: program lokalny

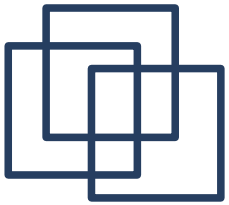
```
            case 'i':
                i = insert(cmdline + 2);
                printf("wstawiono '%s' (%d)\n", cmdline + 2, i);
                break;
            case 'd':
                i = delete(cmdline + 2);
                printf("skasowano '%s' (%d)\n", cmdline + 2, i);
                break;
            case 'l':
                if((i = lookup(cmdline + 2))>=0){
                    printf("slowo '%s' odnalezione (%d)\n", cmdline + 2, i);
                }else{
                    printf("slowo '%s' nieodnalezione (%d)\n", cmdline + 2, i);
                }
                break;
            case 'q':
                    printf("koniec pracy\n");
                    exit(1);
            default:
                    printf("nieznana komenda '%c' \n", cmd);
                    break;
        }// switch
    }// while
}
```



Krok 1: program lokalny

Procedury realizujące poszczególne funkcje programu:

```
#define SETSIZE 100
char wordset[SETSIZE][WORDLEN+1];
int size = 0;

int init(){
    size = 0;
    return 1;
}
int insert(char *word){
    strcpy(wordset[size], word);
    return ++size;
}
int lookup(char *word){
    int i, found = -1;
    for(i=0; i<size; i++){
        if (strcmp(wordset[i], word)==0){
            found = i;
    break;
        }
    }
    return found;
}
```

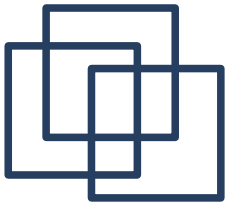


Krok 1: program lokalny

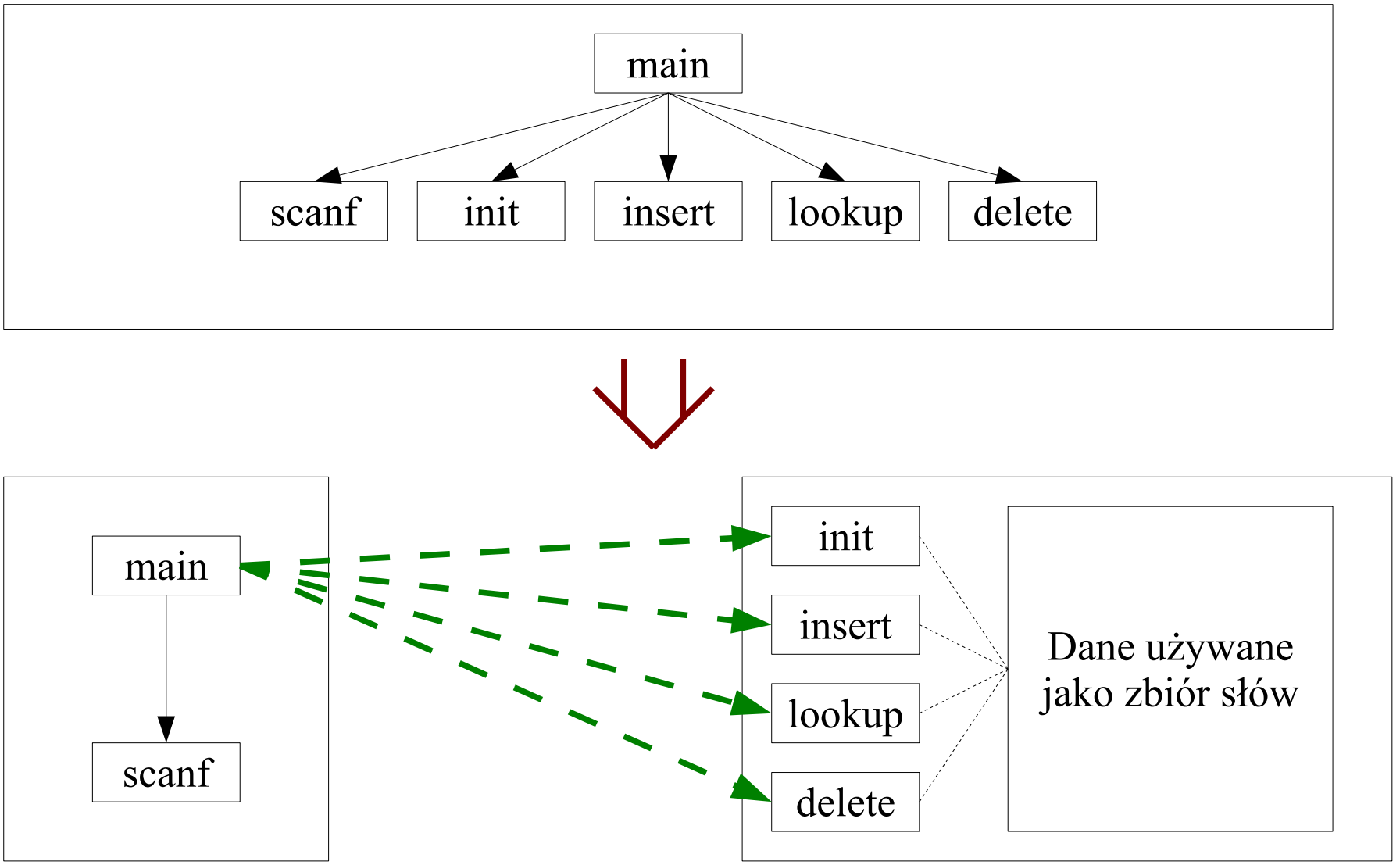
```
int delete(char *word){
    int i, j;
    i = lookup(word);
    if(i>=0){
        for (j=i; j<size; j++){
            strcpy(wordset[j], wordset[j+1]);
        }
        size--;
    }
    return size;
}
```

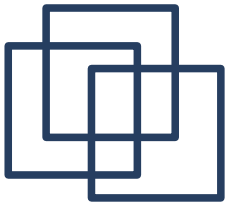
Przykładowe komendy wydawane programowi po uruchomieniu:

```
I
i:jeden
i:dwa
l:jeden
d:jeden
l:jeden
l:dwa
```



Krok 2: podział programu

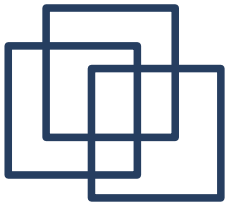




Krok 3: specyfikacja dla rpcgen

Specyfikacja musi zawierać deklaracje: stałych używanych w programie, typów danych, używanych zdalnych programów i funkcji. Specyfikacja jest pisana w języku RPC – plik **`rset.x`**.

```
const WORDLEN=50;
const SETSIZE=100;

struct example{ // przykład, obrazujące struktury XDR
    int id;
    char c;
};
program RSETPROG{                   // deklaracja nazwy zdalnego programu
    version RSETVERS {              // deklaracja wersji
        int INIT(void) = 1;         // deklaracje zdalnych procedur
        int INSERT(string) = 2;
        int LOOKUP(string) = 3;
        int DELETE(string) = 4;
    } = 1;                          // numer wersji
} = 0x22334455;                     // numer programu
```

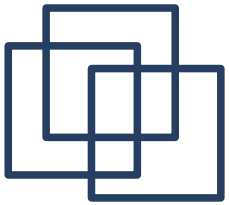


Numeracja zdalnych programów

Przestrzeń numerów dla zdalnych programów:

```
0        - 1fffffff  określone przez firmę Sun
20000000 - 3fffffff  do wykorzystania lokalnego
40000000 - 5fffffff  dla aplikacji wykorzystujących
                     dynamiczne numerowanie programów
60000000 - 7fffffff  pozostałe adresy są zarezerwowane
80000000 - 9fffffff  do późniejszego wykorzystania
a0000000 - bfffffff  i nie powinny być używane
c0000000 - dfffffff  w programach
e0000000 - ffffffff
```

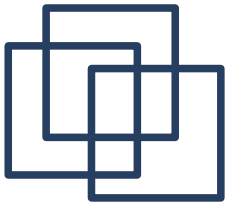

25



Krok 4: użycie rpcgen

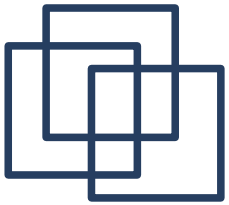
Program rpcgen na podstawie specyfikacji tworzy pliki źródłowe wykorzystywane do kompilacji programu klienckiego i serwera. Plik **rset.h** zawiera deklaracje używane przez oba programy:deklaracje używane przez oba programy:

```
#ifndef _RSET_H_RPCGEN
#define _RSET_H_RPCGEN
#include <rpc/rpc.h>
#ifdef __cplusplus
extern "C" {
#endif

#define WORDLEN 50
#define SETSIZE 100

struct example {
    int id;
    char c;
};
typedef struct example example;
```



Krok 4: użycie rpcgen

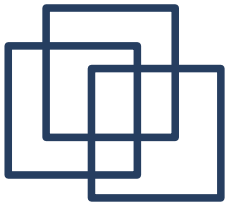
```
#define RSETPROG 0x22334455
#define RSETVERS 1

...

#define INIT 1
extern  int * init_1(void *, CLIENT *);
extern  int * init_1_svc(void *, struct svc_req *);

#define INSERT 2
extern  int * insert_1(char **, CLIENT *);
extern  int * insert_1_svc(char **, struct svc_req *);

#define LOOKUP 3
extern  int * lookup_1(char **, CLIENT *);
extern  int * lookup_1_svc(char **, struct svc_req *);

#define DELETE 4
extern  int * delete_1(char **, CLIENT *);
extern  int * delete_1_svc(char **, struct svc_req *);
extern int rsetprog_1_freeresult (SVCXPRT *, xdrproc_t, caddr_t);
```



Krok 4: użycie rpcgen

```
/* the xdr functions */

extern  bool_t xdr_example (XDR *, example*);

...

#endif /* !_RSET_H_RPCGEN */
```

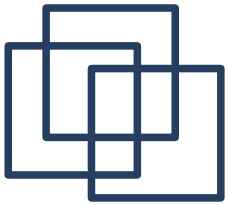
Zdefiniowane procedury zewnętrzne (np. **`init_1()`** i **`init_1_svc()`**) są procedurami łącznikowymi, które powinny być napisane przez programistę.



Krok 4: użycie rpcgen

Plik **`rset_xdr.c`** jest tworzony gdy w specyfikacji były zdefiniowane dodatkowe struktury, z których korzysta program.

```
#include "rset.h"

bool_t
xdr_example (XDR *xdrs, example *objp){
    register int32_t *buf;

     if (!xdr_int (xdrs, &objp->id))
         return FALSE;
     if (!xdr_char (xdrs, &objp->c))
         return FALSE;
    return TRUE;
}
```



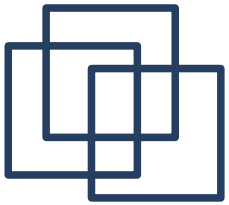
Krok 5: procedury łącznikowe

Plik **rset_clnt.c** zawiera komunikacyjne procedury łącznikowe klienta, np.

```
int *insert_1(char **argp, CLIENT *clnt){
    static int clnt_res;

    memset((char *)&clnt_res, 0, sizeof(clnt_res));
    if (clnt_call (clnt, INSERT,
        (xdrproc_t) xdr_wrapstring, (caddr_t) argp,
        (xdrproc_t) xdr_int, (caddr_t) &clnt_res,
        TIMEOUT) != RPC_SUCCESS) { // wywołanie zdalnej procedury
        return (NULL);
    }
    return (&clnt_res);
}
```

Program klienta uzupełniamy o definicję procedur łącznikowych. Zwykle definiuje się je w osobnym pliku **rset_cif.c** (krok 5).

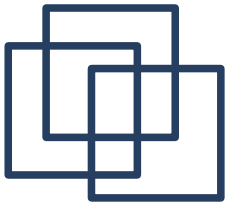
```
int insert(char *word){
    char **arg;
    arg = &word;
    return *insert_1(arg, handle);
}
```



Krok 5: procedury łącznikowe

Plik **rset_svc.c** zawiera kod serwera.

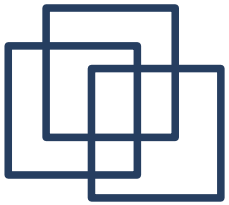
```
...
char *result;
xdrproc_t _xdr_argument, _xdr_result;
char *(*local)(char *, struct svc_req *);
...
    case INSERT:
        _xdr_argument = (xdrproc_t) xdr_wrapstring;
        _xdr_result = (xdrproc_t) xdr_int;
        local = (char *(*)(char *, struct svc_req *)) insert_1_svc;
        break;
...
memset ((char *)&argument, 0, sizeof (argument));
if (!svc_getargs (transp, (xdrproc_t)_xdr_argument, (caddr_t)&argument)){
    svcerr_decode (transp);
    return;
} // pobranie argumentów
result = (*local)((char *)&argument, rqstp); // wywołanie procedury
if (result != NULL && !svc_sendreply(transp, (xdrproc_t)_xdr_result, result)){
    svcerr_systemerr (transp);
} // wysłanie odpowiedzi
if (!svc_freeargs (transp, (xdrproc_t) _xdr_argument, (caddr_t) &argument)){
    fprintf (stderr, "%s", "unable to free arguments"); exit(1);
} // zwolnienie pamięci
```



Krok 5: procedury łącznikowe

Procedury łącznikowe serwera zazwyczaj grupowane są w osobnym pliku (**rset_sif.c**).

```
#include<rpc/rpc.h>
#include "rset.h"

static int retint;

int *init_1_svc(void *w, struct svc_req *sreq){
    retint = init();
    return &retint;
}

int *insert_1_svc(char **w, struct svc_req *sreq){
    retint = insert(*w);
    return &retint;
}
...
```



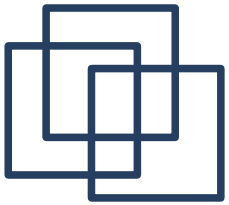
Krok 6: kompilacja klienta

W pliku źródłowym klienta **`rset.c`** należy dodać fragmenty kodu związane z inicjacją połączenia RPC.

```
#include<rpc/rpc.h>
#include"rset.h"
#define RMACHINE "komputer.domena"   // nazwa lub adres ip serwera
CLIENT *handle;                      // uchwyt wykorzystywany do komunikacji
...
if ((handle=clnt_create(RMACHINE, RSETPROG, RSETVERS, "tcp"))==NULL){
    printf("nie mozna nawiazac polaczenia\n");
    exit(1);
} // nawiązanie połączenia
```

Kompilacja i linkowanie:

```
gcc -o rset rset.c rset_cif.c rset_clnt.c rset_xdr.c
```

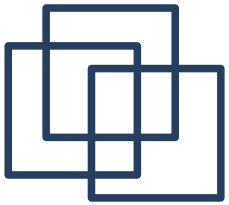


Krok 7: kompilacja serwera

Na serwer, oprócz plików wygenerowanych automatycznie: **`rset_svr.c`**, **`rset_xdr.c`** i funkcji łącznikowych **`rset_sif.c`** i składa się plik zawierający implementację zdalnych funkcji **`rset_fun.c`**.

Kompilacja i linkowanie:

```
gcc -o rsetd rset_svr.c rset_fun.c rset_sif.c rset_xdr.c
```



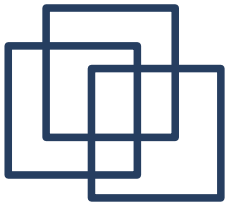
Krok 8: testowanie i uruchomienie programu

Serwer powinien rozpocząć działanie zanim klient nawiąże próbę połączenia.

Zarówno program klienta jak i serwer można w celach testowych uruchomić na tym samym komputerze. W tym celu należy w programie **`rset.c`** zastąpić istniejącą deklarację przez:

```
#define RMACHINE "localhost"
```

Aby ułatwić zarządzanie projektem podczas poprawiania lub wprowadzania zmian, warto zautomatyzować proces kompilacji za pomocą narzędzia **`make`**.



Podsumowanie

Zwykle do konstrukcji programów rozproszonych wykorzystywane jest narzędzie rpcgen. Proces budowania aplikacji rozproszonej można podzielić na osiem etapów. Najpierw programista buduje zwykły program lokalny. Następnie musi on zostać podzielony na część lokalną i zdalną. Program rpcgen automatycznie generuje większość kodu potrzebnego do obsługi komunikacji RPC. Pomimo tego konwersją do modelu rozproszonego zwykle jest pracochłonna i wymaga uwagi.