RPC – ZDALNE WYWOŁYWANIE PROCEDUR (*ang. Remote Procedure Calls*)

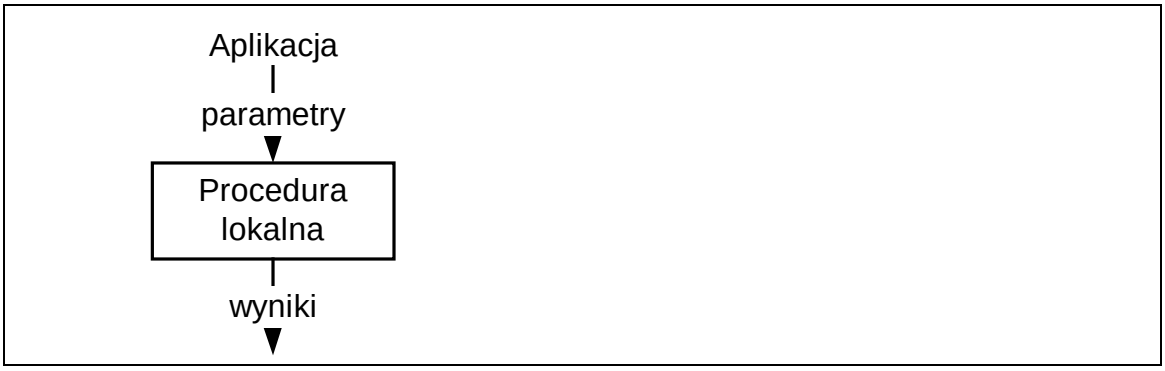
1 Informacje wstępne

1.1 Koncepcja

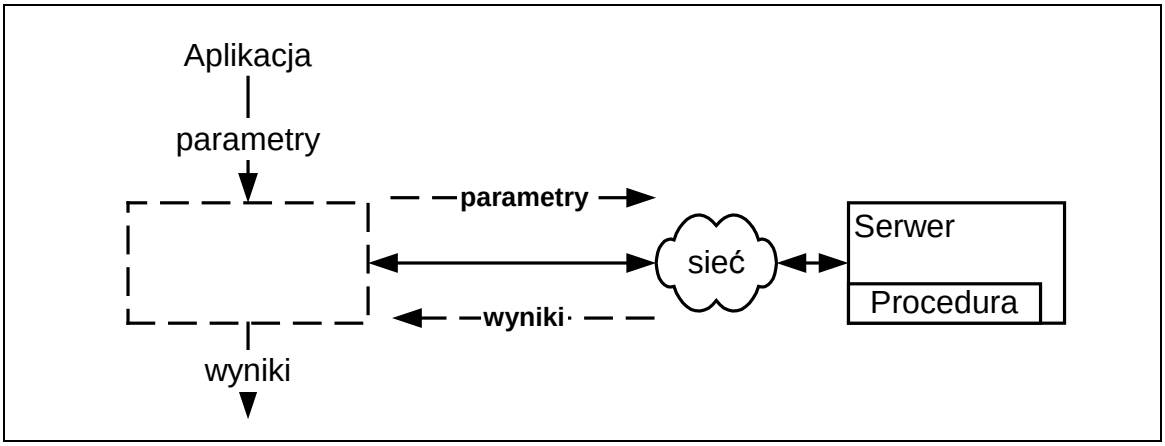
Model klient serwer pozwala na rozwiązanie szerokiej klasy problemów posiada jednak ograniczenia:

- Odwołanie się do wejścia / wyjścia (receive / send)
- Występuje problem reprezentacji danych (w systemach heterogenicznych)
- Model zorientowany na dane (przesyłanie danych zamiast wykonywanie akcji)

W 1984 Birell i Nelson zaproponowali organizację przetwarzania w środowiskach współbieżnych i rozproszonych za pomocą paradygmatu wywoływania zdalnych procedur. Idea jest elegancka ale też powoduje problemy.



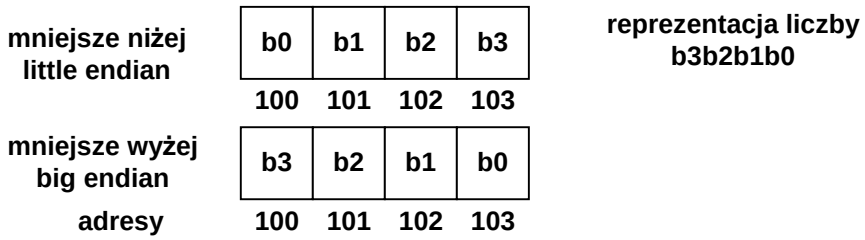
Rys. 1-1 Lokalne wywołanie procedury



Rys. 1-2 Zdalne wywołanie procedury

1.2 Przekazywanie parametrów

Maszyny mogą się różnić sposobem reprezentacji danych (big endian, little endian, ASCII, EBDIC, Unicode). Stąd konieczność konwersji do postaci kanonicznej.



Różne reprezentacje liczby int

Przetaczanie parametrów (ang. *parameters marschalling*) – pakowanie parametrów procedury do komunikatu z jednoczesną konwersją danych.

Przetaczanie parametrów obejmuje:
1. Konwersję formatu komunikatu
2. Serializację danych

W języku C stosowane są dwa rodzaje przekazywania parametrów:
1. Przez wartość – parametry bezpośrednio kopiowane są na stos
2. Przez odniesienie – na stos kopiowany jest adres parametru

Przykład:

```
int read(int fh, char * bufor, int size)
```

Trudności w przekazywaniu parametrów:

- Definicja procedury powinna określić które parametry są wejściowe, które wyjściowe a które przekazują dane w obydwu kierunkach. W języku C parametry przekazywane przez odniesienie tego nie specyfikują.
- Procedura wywołująca i wywoływana działają na różnych maszynach w różnych przestrzeniach adresowych. Występuje problem z użyciem wskaźników. Wskaźnik ma znaczenie tylko w obrębie procesu w której został utworzony.
- Do przekazywania parametrów nie mogą być wykorzystane zmienne globalne

Uwaga !
Należy poinformować system jakiego typu są parametry i czy są one wejściowe czy wyjściowe. Jest to robione w języku opisu interfejsu IDL (ang. *Interface Description Language*).

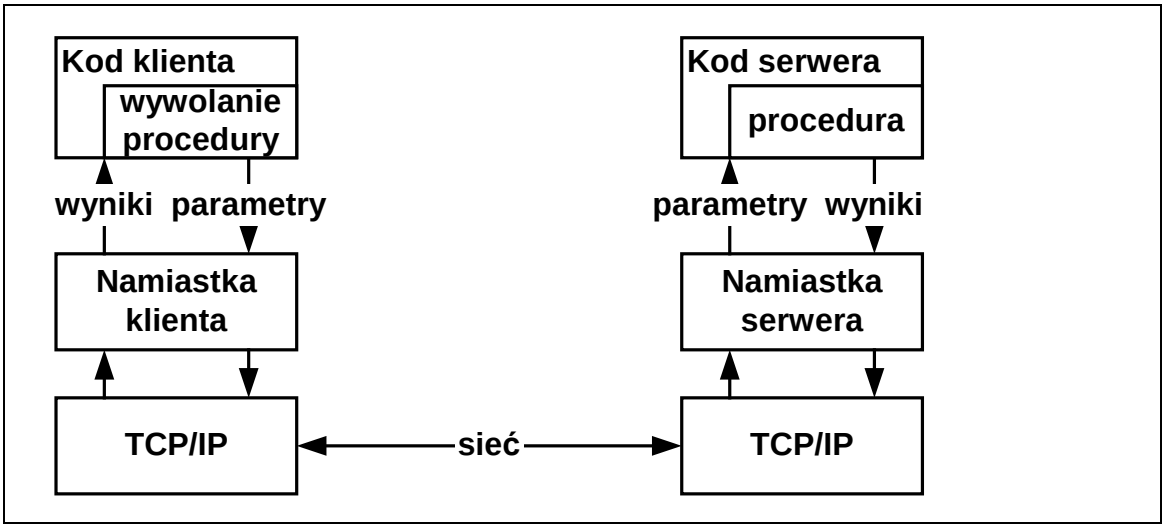
Mimo że można sobie poradzić z prostymi wskaźnikami w RPC trudno przekazać informacje o strukturach danych opartych na wskaźnikach (listy, grafy).

1.3 Budowa aplikacji RPC

Łącznikami aplikacji klienta i serwera są:

- Namiastka klienta (*ang. client stub)* - reprezentuje serwer po stronie klienta
- Namiastka serwera (*ang. server stub*) - reprezentuje klienta po stronie serwera

Wywołanie procedury czytaj zastąpione wywołaniem namiastki klienta.



Rys. 1-3 Przebieg zdalnego wywołania procedury

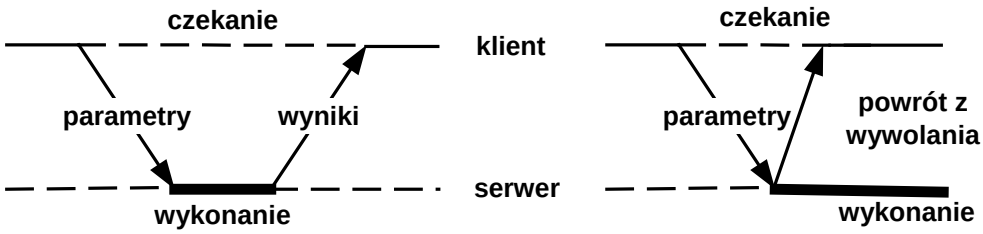
Zdalne wywołanie procedury odbywa się w krokach:

1. Aplikacja klienta wywołuje namiastkę klienta
2. Namiastka klienta buduje komunikat i przechodzi do jądra OS
3. Jądro przesyła komunikat do jądra maszyny odległej
4. Namiastka serwera rozpakowuje parametry i wywołuje serwer
5. Serwer wykonuje zdalną procedurę i zwraca wynik stopce serwera
6. Namiastka pakuje wyniki w komunikat i przesyła do maszyny klienta
7. Jądro klienta przekazuje komunikat stopce klienta
8. Namiastka klienta rozpakowuje wynik i zwraca go klientowi

Synchroniczne i asynchroniczne RPC

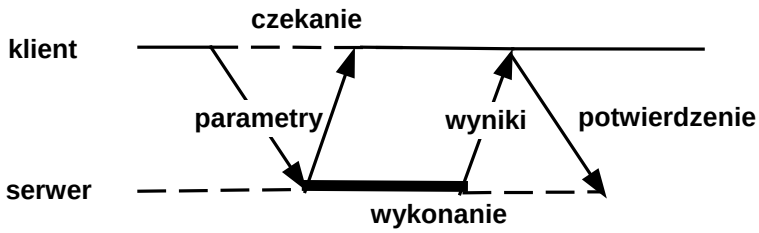
Rodzaje RPC ze względu na synchronizacje:

- Synchroniczne RPC - klient czeka na odpowiedź serwera
- Asynchroniczne RPC - klient przekazuje parametry do serwera i kontynuuje działanie.
- Odroczone asynchroniczne RPC - klient przekazuje parametry do serwera i kontynuuje działanie. Gdy serwer opracuje odpowiedź wywołuje procedurę po stronie klienta.



Synchroniczne RPC **Asynchroniczne RPC**

Synchroniczne i asynchroniczne RPC



Odroczone asynchroniczne RPC

Odroczone asynchroniczne RPC

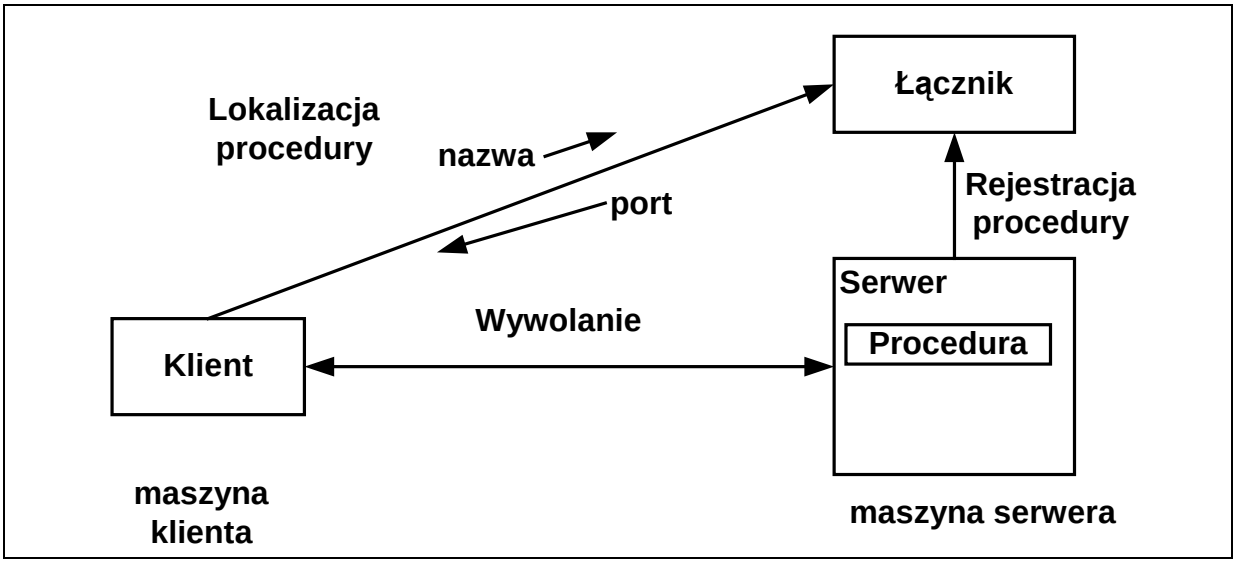
2. Wiązanie dynamiczne

Przy zdalnym wywoływaniu procedur powstaje pytanie jak klient ma zlokalizować procedury serwera.

Wiązanie (*ang. binding*) – odwzorowanie nazwy (procedury RPC) w konkretny obiekt określony identyfikatorem komunikacyjnym. Postać identyfikatora zależy od systemu (np. adres gniazdka - IP, port).

Łącznik (*ang. binder*) – specjalna usługa RPC utrzymująca tablicę odwzorowań nazw usług (procedur RPC) na porty serwerów tych usług.

Łącznik utrzymywany jest przez serwery które udostępniają identyfikatory portów swoim klientom.



Rys. 2-1 Rola łącznika w RPC

Wywołanie	Wejście	Wyjście
Rejestracja	*Nazwa, wersja, identyfikator komunikacyjny*	-
Wyrejestrowanie	*Nazwa, wersja*	-
Lokalizacja	*Nazwa, wersja*	*identyfikator komunikacyjny*

Funkcje interfejsowe łącznika

Od usług łącznika zależą wszystkie inne usługi. Dlatego łączniki tworzy się tak aby tolerowały awarie. Np. tablice odwzorowań zapisuje się w pliku z którego mogą być one wczytane w przypadku awarii.

Lokalizowanie łącznika

Zanim klient RPC zrobi cokolwiek musi skontaktować się z łącznikiem. Skąd ma znać jego lokalizację? Stosowane są rozwiązania:

1. Łącznik działa na komputerze którego adres jest dobrze znany. Gdy jego lokalizacja się zmieni wymagana jest rekompilacja klientów.
2. Za dostarczanie aktualnego adresu łącznika odpowiada system operacyjny komputera klienta i serwera. Można go utrzymywać w postaci zmiennej środowiskowej.
3. Rozpoczynając swoje działanie programy klienta lub serwera lokalizuje łącznik za pomocą rozgłaszania. Komunikat rozgłaszający zawierać może numer portu łącznika a łącznik odpowiada adresem IP komputera na którym się znajduje.

Obszary problemów w RPC:

Brak przeźroczystości

Wywołanie zdalnej procedury powinno być przeźroczyste to znaczy wywołanie lokalne i zdalne powinno mieć jednakową postać. W rzeczywistych implementacjach tak nie jest.

Problem zmiennych globalnych

Gdy jedna z procedur jest lokalna a druga zdalna to nie mogą korzystać one ze zmiennych globalnych.

Problem z pewnymi typami danych wejściowych

- Tablice o nieokreślonej długości
- Wskaźniki

3. Tworzenie aplikacji RPC w Sun RPC

3.1 Poziomy tworzenia aplikacji RPC

Realizacja Sun RPC oparta jest na gniazdkach. Mechanizm gniazdek jest zamaskowany przed użytkownikiem. Ma on do dyspozycji funkcje wyższego poziomu.

Można korzystać z protokołu:
- TCP
- UDP (rozmiar danych ograniczony do 8 KB).

W skład RPC wchodzą:

- Biblioteki funkcji
- Narzędzie **`rpcgen`**

Semantyka wywołań - wykonanie najwyżej raz.

Dostępne trzy poziomy:

1. Poziom pierwszy - gotowych funkcji (np. **`nusers`** - liczba zal. użytk.)
2. Poziom pośredni – łatwy w użyciu lecz ograniczona funkcjonalność
3. Poziom trzeci – stosowane są funkcje niskiego poziomu, pełna funkcjonalność

Poziom pośredni

Najczęściej stosowany jest poziom pośredni. Jest on odpowiedni dla większości typowych aplikacji.

Aplikacja poziomu pośredniego nie umożliwia:

- Kontroli przeterminowań
- Użycie wielu procesów / wątków po stronie serwera
- Elastycznej obsługi błędów
- Użycia zaawansowanej identyfikacji strony wywołującej

Tworzenie aplikacji RPC odbywa się w następujących krokach:

1 Utworzenie interfejsu serwera – specyfikacja (w języku opisu interfejsu RPCGEN) co serwer ma wykonać i jak przekazuje się parametry.
2 Przy użyciu programu **`rpcgen`** generuje się namiastkę klienta (*ang. klient stub*), namiastkę serwera (*ang. server stub*), plik nagłówkowy, plik konwersji.
3 Implementacja usług serwerowych.
4 Implementacja aplikacji klienta – wykorzystanie stopki klienta.

Program serwera działa według schematu:

1. Otrzymanie identyfikatora transportu (*ang. transport handle*)
2. Zarejestrowanie usługi u demona **`portmap`**
3. Oczekiwanie na zgłoszenia klienta i wykonywanie jego zleceń

Program klienta działa według schematu:

1 Otrzymanie identyfikatora klienta (*ang. client handle*)
2 Wywoływanie odległych procedur
3 Likwidacja identyfikatora klienta gdy nie jest potrzebny

3.2 Język opisu interfejsu RPCGEN

RPCGEN jest językiem definiowania interfejsu i prekompilatorem. Na podstawie definicji interfejsu RPCGEN tworzy następujące pliki w języku C:

1. Namiastka klienta (*ang. client stub*)
2. Namiastka serwera (*ang. server stub*)
3. Plik konwersji danych
4. Plik nagłówkowy

Język RPCGEN akceptuje następujące typy danych:

Typ pusty	**`void`**
Znak	**`char`**
Typ całkowity	**`int, short int, long int`** (**`unsigned`**)
Typ zmiennoprzecinkowy	**`float, double`**
Tablice o stałej długości	**`typ nazwa [zakres]`**
Tablice o zmiennej długości	**`typ nazwa <zakres>`**
Łańcuch	**`string <zakres>`**
Struktura	**`struct { ....}`**
Typ złożony	typedef nazwa definicja

Nie jest możliwe użycie wskaźnika do wskaźnika. Można w takim przypadku należy użyć wskaźnika do typu złożonego.

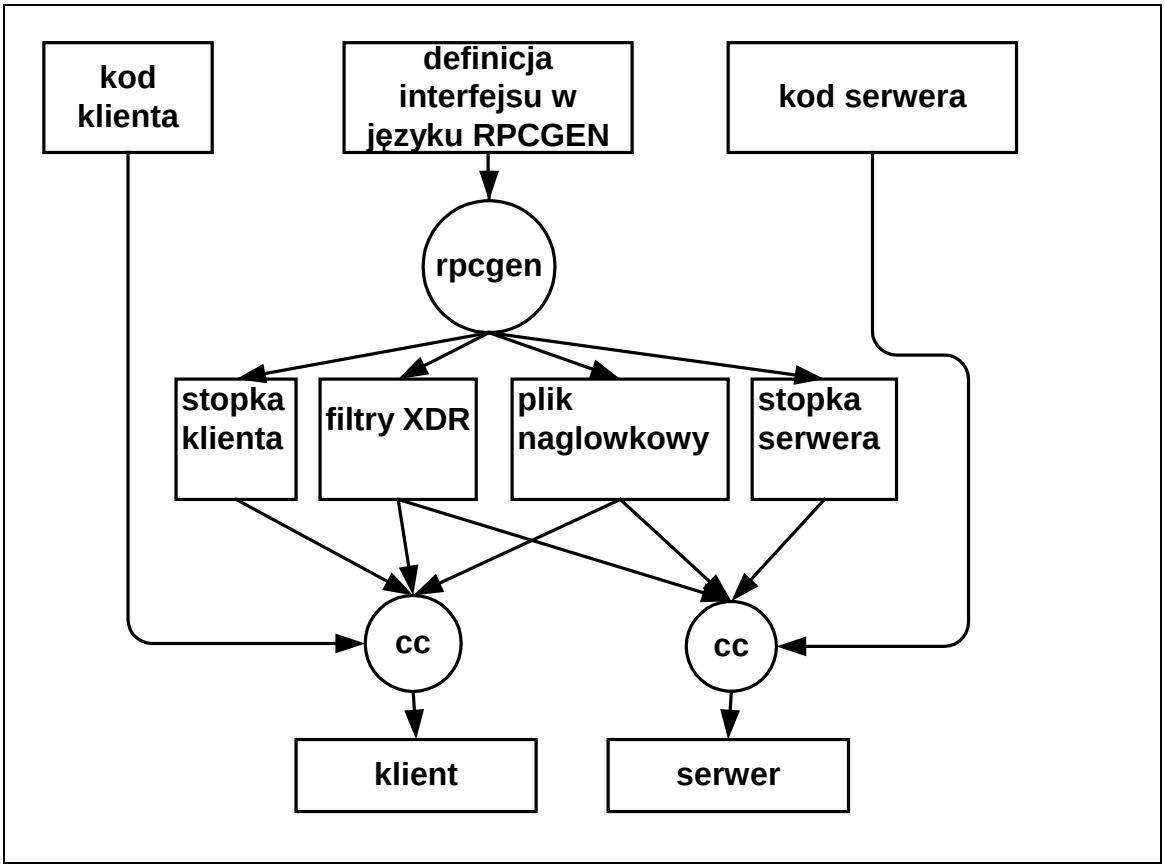
Przekazywanie argumentów:

1. Wywołanie odległej procedury dopuszcza tylko jeden argument wywołania i zwraca jeden wynik.
2. Gdy występuje więcej elementów to należy umieścić je w strukturach.
3. W programach klienta i serwera argumentem lub wynikiem jest wtedy wskaźnik na strukturę.

3.3 Generowanie aplikacji RPC

Aby utworzyć aplikację RPC należy:

1. Utworzyć w języku RPCGEN plik interfejsu opisujący funkcje które mają być zaimplementowane. Nadać numery programu, wersji i funkcji.
2. Skompilować plik interfejsu za pomocą programu **rpcgen**. W rezultacie utworzony zostanie plik nagłówkowy, plik filtrów XDR, namiastka klienta i namiastka serwera.
3. Napisać program klienta korzystając z definicji funkcji klienta podanych w pliku nagłówkowym.
4. Utworzyć plik serwera korzystając z definicji funkcji serwera podanych w pliku nagłówkowym. Dokonać implementacji tych funkcji.
5. Skompilować plik klienta, stopki klienta, filtrów XDR i połączyć w program klienta.
6. Skompilować plik serwera, stopki serwera, filtrów XDR i połączyć w program serwera.



Rys. 3-1 Tworzenie aplikacji przy pomocy prekompilatora RPCGEN

Przykład: sumowanie dwóch liczb

```
struct integers{
  int x1;
  int x2;
};

typedef struct integers intargs;


program PROG3 {
  version WERS1 {
    int sumuj(intargs) = 1;
    } = 1;
} =  0x30000005;
```

Interfejs **suma.x** – sumowanie dwóch liczbw języku opisu interfejsu

Specyfikacja interfejsu zawiera:

1. Numer programu – 32 bitowa liczba całkowita unikalna w ramach systemu i spełniająca podane niżej ograniczenia.
2. Numer wersji programu – liczba 32 bitowa dodatnia unikalna w ramach programu
3. Numer funkcji - liczba 32 bitowa unikalna w ramach wersji

Numer HEX	Opis
00000000 – 1FFFFFFF	Sun
20000000 - 3FFFFFF	Administrator lokalny
40000000 - 5FFFFFF	Programista
60000000 - FFFFFFF	Zastrzeżone

Numery programów w Sun RPC

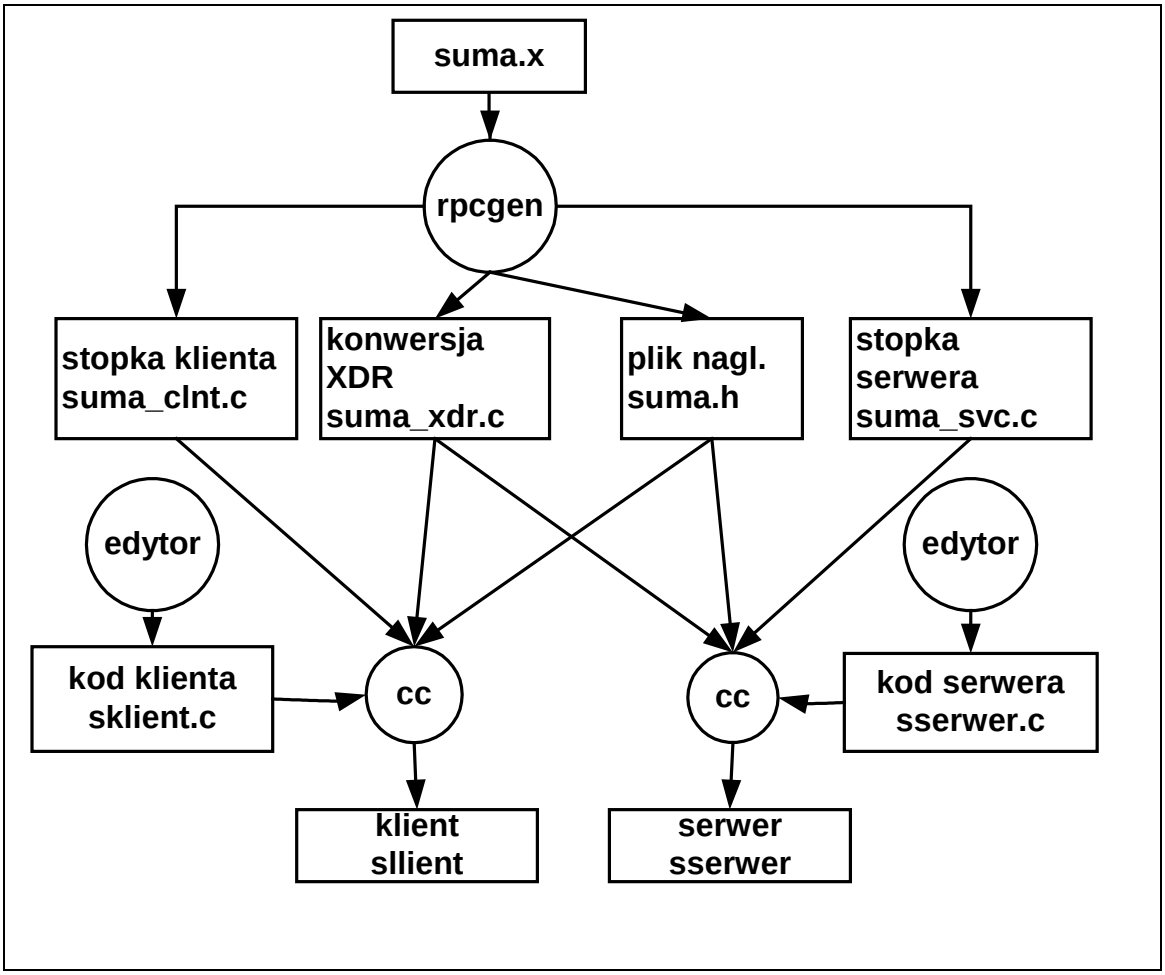
Generowanie stopki klienta i serwera:

$ rpcgen suma

Wynik:

suma_clnt.c – Namiastka klienta (ang. client stub)
suma_svc.c – Namiastka serwera (ang. server stub)
suma_xdr.c - plik konwersji danych
suma.h - plik nagłówkowy

Należy dopisać:
sserwer.c – plik serwera
sklient.c – plik klienta



Rys. 3-2 Przykład tworzenia aplikacji RPC

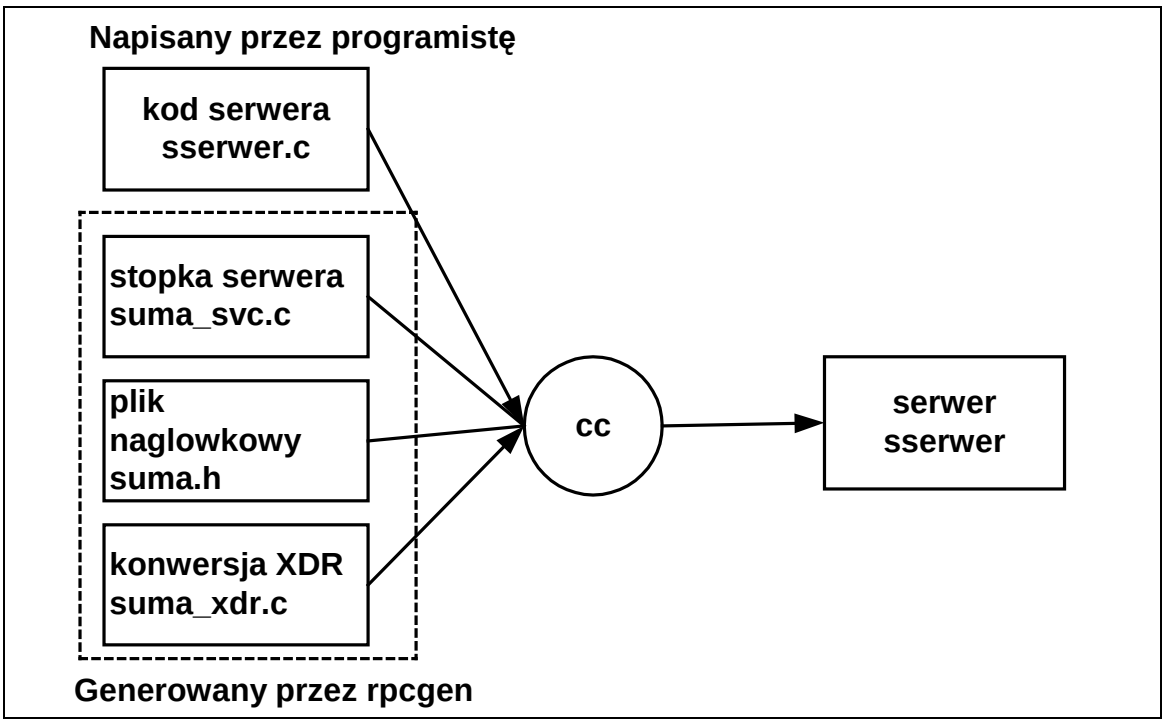
```
// Definicja danych
struct integers {
    int x1;
    int x2;
};

typedef struct integers integers;
// Definicje funkcji klienta
extern  int * sumuj_1(intargs *, CLIENT *);

// Definicje funkcji serwera
extern int* sumuj_1_svc(intargs *, struct svc_req *);
```

Fragment pliku suma.h

3.4 Tworzenie serwera

Aby utworzyć serwer należy:

1. Utworzyć plik z kodem serwera **`sserwer.c`**
2. Zaimplementować w nim funkcję serwera (sumuj_**`1_svc)`** której prototyp podany jest w pliku nagłówkowym.
3. Skompilować pliki i połączyć w plik wykonywalny **`sserwer`**



Rys. 3-3 Przykład tworzenia serwera w RPC

```
// Program sumowania dwu liczb - sserwer
// Kompilacja: gcc sserwer.c suma_svc.c suma_xdr.c -o
sserwer -lrpc

#include "suma.h"

int *sumuj_1_svc(intargs* arg, struct svc_req * s) {
  static int result;
   result = arg->x1 + arg->x2;
   return &result;
}
```

Przykład 3-1 Kod programu serwera

3.5 Tworzenie klienta:

Aby utworzyć kod klienta należy znać:
1. Interfejs serwera – plik **`suma.x`**
2. Nazwę sieciową komputera serwera

Do wygenerowanego przez **`rpcgen`** stopki klienta dopisuje się własny kod klienta zawierający funkcję **`main()`**.

Funkcje po stronie klienta:

Tworzenie identyfikatora klienta:

```
CLIENT * clnt_create(char *host, int prognum, int
progver, char *protocol)
```

host – nazwa serwera
prognum – numer programu
progver – numer wersji
protocol – typ protokołu („udp” lub „tcp”)

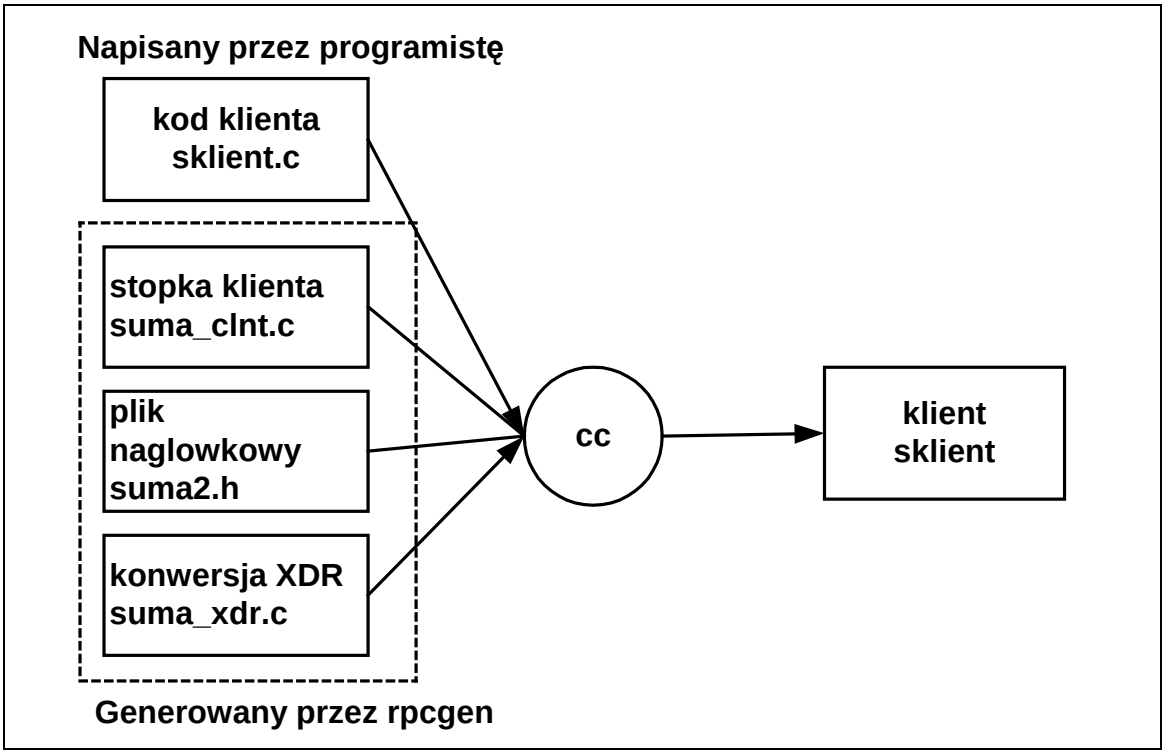
Przykład:
```
cli = clnt_create(HOST, PROG1, VER1, "tcp")
```

Wywołanie zdalnej procedury:

```
enum clnt clnt_call(CLIENT *clnt , int procnum,
xdrproc_t inproc, char *in, xdrproc_t outproc,
char *out, struct timeval timeout)
```

clnt – identyfikator klienta
procnum – numer procedury
inproc – procedura kodująca XDR
in – bufor na dane wejściowe
outproc – procedura dekodująca XDR
out – bufor na zwracane wyniki
timeout – limit czasowy (domyślnie 25 sekund)

Funkcja powoduje wywołanie procedury zdalnej o numerze **`procnum`**.



Rys. 3-4 Tworzenie klienta RPC

```
// Kod programu klienta
// Program sumowania dwu liczb
// Nalezy uruchomic program ssuma
// Kompilacja: gcc sklient.c suma_clnt.c suma_xdr.c -
o sklient -lrpc
#include "suma.h"
#include <stdio.h>
#include <stdlib.h>

#define HOST "192.168.1.222"


int main(int argc, char *argv[]) {
  CLIENT *c1;
  integers arg;
  int *sum;
  char* host = "localhost";
  if(argc < 3) {
     printf("Uzycie: ssuma l1 l2\n");
     exit(0);
  }
  c1 = clnt_create(host, PROG3, WERS1, "tcp");
  if (c1 == NULL) {
     clnt_pcreateerror(host);
     return -1;
  }
  arg.x1 = atoi(argv[1]);
  arg.x2 = atoi(argv[2]);

  sum = (int*) sumuj_1(&arg, c1);
  if(sum == NULL) {
    clnt_perror(c1, "blad wywolania RPC");
    return -1;
  }
  printf(" %d + %d = %d \n",arg.x1,arg.x2,*sum);
  clnt_destroy(c1);
  return 0;
}
```

Przykład 3-2 Kod programu klienta

3.6 Komunikacja między klientem a serwerem

Komunikacja pomiędzy klientem a serwerem odbywa się za pomocą protokołów TCP lub UDP.

Adresowanie w RPC:
- Nazwa (adres IP) komputera na którym uruchomiony jest serwer
- Numer programu
- Numer wersji
- Numer procedury

Aby móc skorzystać z RPC należy uruchomić program łącznika **`portmap`**.

Procedury zdalne na danym komputerze są identyfikowane przez trzy liczby:
- numer programu,
- numer wersji programu
- numer procedury.

Numer programu i wersji – identyfikuje procesy serwerowe
Numery procedur identyfikują procedury w serwerze

Portmapper RPC jest serwerem nazewniczym , który zamienia numery programu RPC na numery portów protokołu TCP albo UDP. Musi być on uruchomiony, aby móc używać na tej maszynie odwołań RPC do serwerów RPC.

Kiedy serwer RPC jest startowany, poinformuje on portmapper na których portach nasłuchuje, i jakimi numerami programowymi RPC może służyć.

Kiedy klient chce odwołać się przez RPC do danego numeru programu, najpierw skontaktuje się z portmapperem na maszynie serwerowej, aby określić numer portu, do którego należy wysłać pakiety RPC. Komunikacja z portmapperem odbywa się na ustalonym porcie 111.

Aby uzyskać informację o numerach procedur, portach można użyć narzędzia **`rpcinfo`**.

```
$ rpcinfo –p nazwa_serwera
```

```
   program vers proto   port
    100000    4   tcp    111  portmapper
    100000    4   udp    111  portmapper
 805306372    1   udp  65534
 805306373    1   udp  65297

program 805306373 version 1 ready and waiting
```

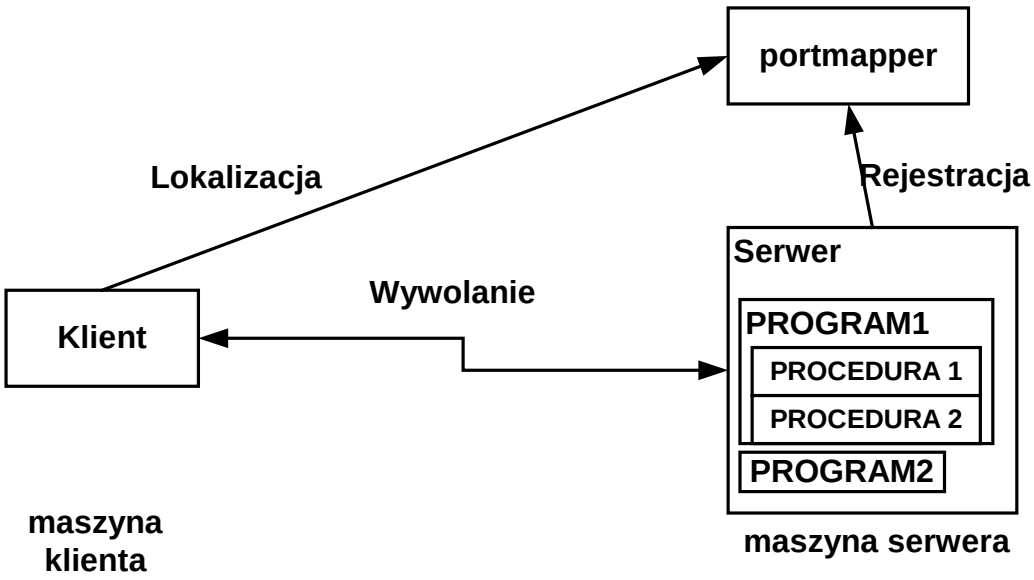
Uruchomienie aplikacji:

Na maszynie serwera (np. *zdalna*):

$ portmap & Uruchomienie łącznika (gdy nie jest uruchomiony)
$sserwer Uruchomienie serwera

Na maszynie klienta:

$ rpcinfo -p *zdalna* Testowanie łącznika
$ sklient Uruchomienie serwera



Rys. 3-5 Schemat rozproszonej aplikacji RPC

4. Cechy systemu RPC:

Usługa RPC implementuje monitor jednak bez możliwości czekania wewnątrz procedur monitora. Brak odpowiednika zmiennej warunkowej.

Zalety:
- Prostota
- Duża wydajność
- Rozpowszechnienie standardu

Wady:
- Brak wsparcia serwerów RPC dla wielowątkowości
- Brak implementacji czekania (blokowanie klienta)
- Słabe techniki autoryzacji klienta
- Brak zabezpieczenia przed konfliktem numerów programów
- W interfejsach tylko funkcje jednoargumentowe
- Trudności w użyciu wskaźników do przekazywania parametrów
- Wsparcie tylko dla języka C

Rozwinięciem idei RPC są nowsze systemy:
- Corba (*ang. Common Object Request Broker Architecture*)
- .net Remoting (MS Windows)
- DCOM (*ang. Distributed Component Object Mode*)
- SOAP (*ang. Simple Object Access Protocol*)
- Java RMI (*ang. Remote Method Invocation*)
- Ninf, NetSolve/GridSolve (gridy)

5. Literatura:

- Michael Gabassi, Bertrand Dupoy, Przetwarzanie rozproszone w systemie UNIX, Lupus 1995.
- G. Colouris, J. Dollimore, Systemy rozproszone WNT 1998.
- Andrew S. Tannenbaum, Systemy rozproszone zasady i paradygmaty, WNT Warszawa 2006.