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

PDF created with pdfFactory Pro trial version www.pdffactory.com

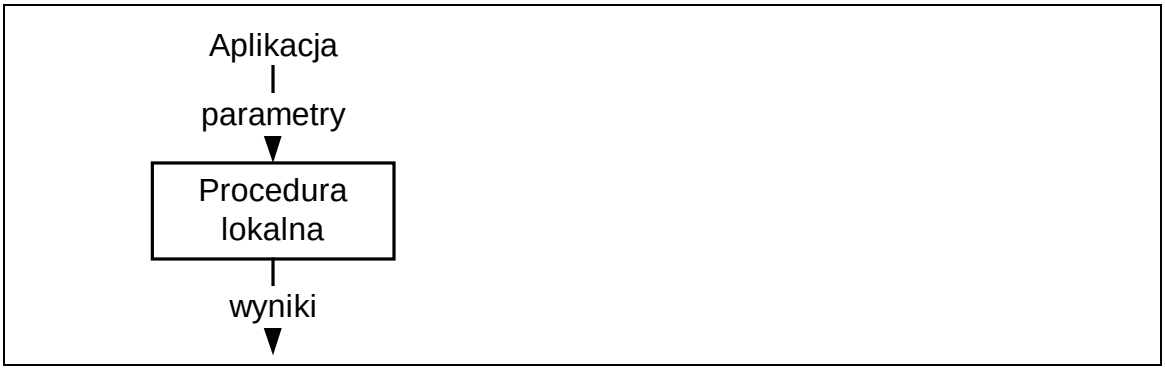
1 Informacje wstępne

1.1 Koncepcja

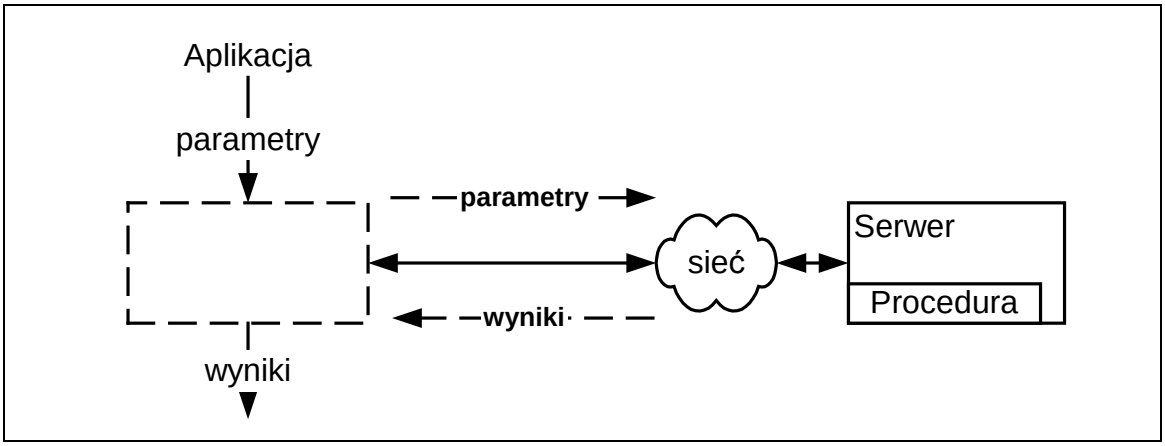
Model klient serwer pozwala na rozwiązanie szerokiej klasy problemów posiada jednak ograniczenia:

- 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

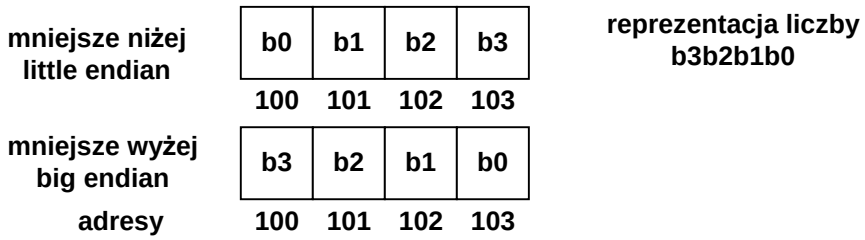


Rys. 1-2 Zdalne wywołanie procedury

PDF created with pdfFactory Pro trial version www.pdffactory.com

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

PDF created with pdfFactory Pro trial version www.pdffactory.com

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*).

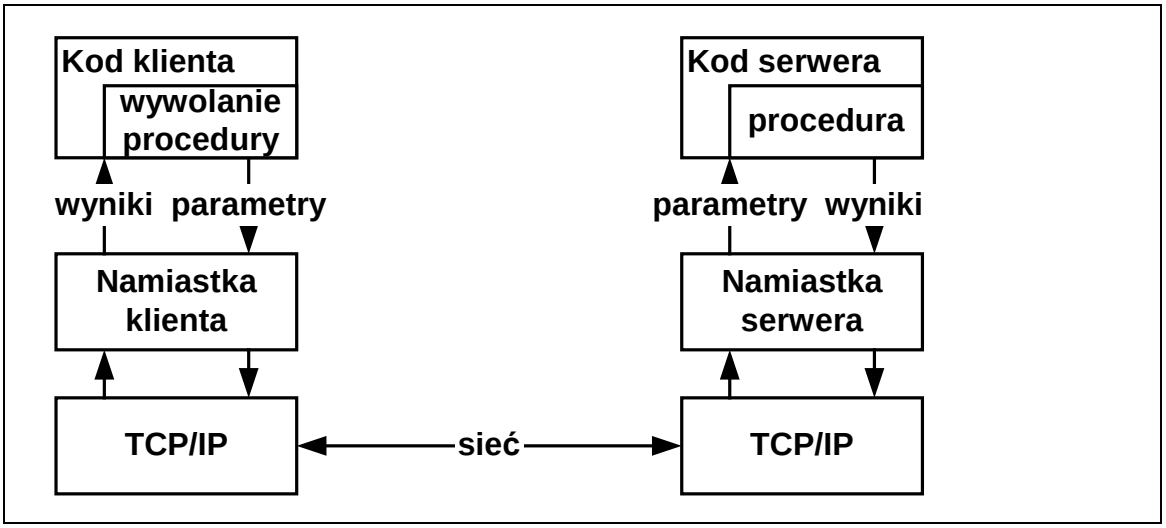
Mimo że można sobie poradzić z prostymi wskaźnikami w RPC trudno przekazać informacje o strukturach danych opartych na wskaźnikach (listy, grafy).

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

PDF created with pdfFactory Pro trial version www.pdffactory.com

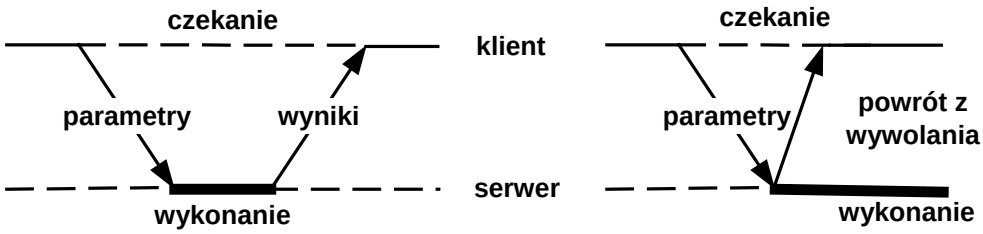
Zdalne wywołanie procedury odbywa się w krokach:

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

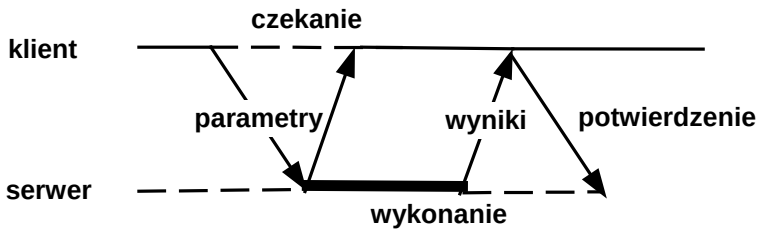
Rodzaje RPC ze względu na synchronizacje:

- 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

PDF created with pdfFactory Pro trial version www.pdffactory.com

2. Obsługa błędów i sytuacji wyjątkowych

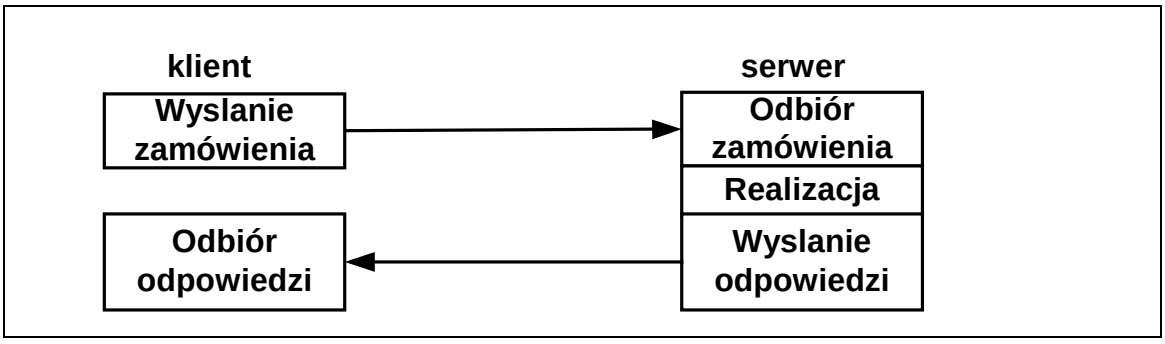
System RPC jest systemem działającym w środowisku sieciowym a zatem należy się liczyć z trudnościami komunikacyjnymi. Ponieważ dowolne wywołanie RPC może się skończyć niepowodzeniem wymagane jest informowanie strony wywołującej o błędach i obsługa sytuacji wyjątkowych.

Obsługa błędów może być oparta o:
- Zwracanie kodu błędów przez procedury
- Obsługę wyjątków (konstrukcje jak np. **`try {...} catch {...}`**, sygnały)

Gdy zdalna procedura ma zwrócić jakiś wynik (a tak się dzieje w implementacji Sun RPC) powstaje problem jak odróżnić ten wynik od kodu powrotu. Należy zastosować mechanizm wyjątku.

Typy awarii RPC:
- Awaria komunikacji
- Awaria serwera
- Awaria klienta

1. Klient nie może zlokalizować serwera
2. Zaginął komunikat zamawiający od klienta do serwera
3. Zaginęła odpowiedź od serwera do klienta
4. Serwer uległ awarii po otrzymaniu zamówienia
5. Klient uległ awarii po wysłaniu zamówienia



Rys. 2-1 Przebieg komunikacji klienta i serwera

PDF created with pdfFactory Pro trial version www.pdffactory.com

2.1 Awarie komunikacji

Zaginął komunikat zamawiający
Po wysłaniu komunikatu należy uruchomić czasomierz. Gdy w zadanym czasie nie przyjdzie odpowiedź następuje retransmisja komunikatu. Problem powstaje gdy komunikat nie zaginął a serwer wykona zamówienie wiele razy. Serwer nie wie czy jest to nowe żądanie czy powtórzenie.

Zaginęła odpowiedź od serwera do klienta
Gdy brak odpowiedzi klient nie wie jaki był tego powód:
- Zaginęło zamówienie
- Serwer działa powoli
- Zaginęła odpowiedź.

Niektóre zamówienia można powtarzać bezpiecznie – akcje idempotentne (np. podanie zawartości pliku).
Innych nie można powtarzać - akcje nie idempotentne (np. zlecenie przelewu z konta na konto).

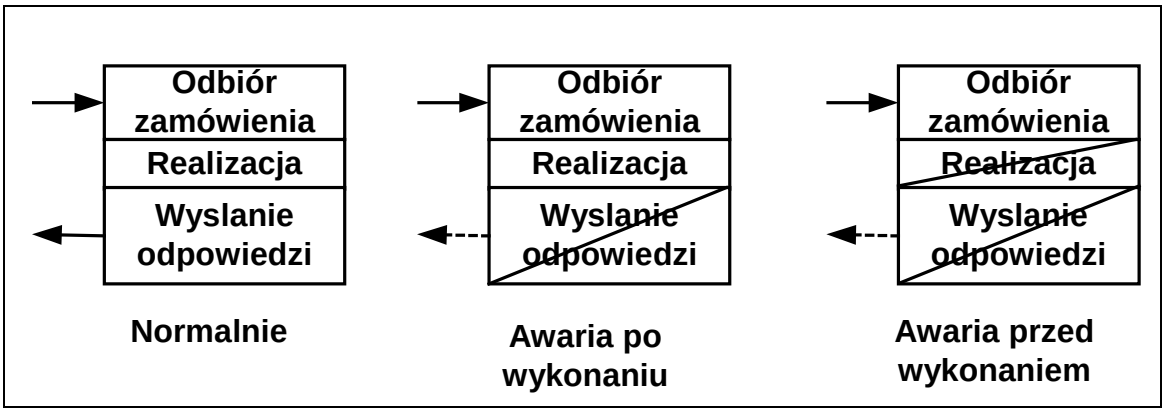
> **Operacja idempotentna** (*ang. idempotent operation*)
> Operacja idempotentna to taka operacja której wielokrotne wykonanie powoduje taki sam skutek jakby była wykonana jednokrotnie.

Rozwiązaniem jest numerowanie filtrowanie zamówień. Polega ono na umieszczeniu w komunikacie informacji czy zamówienie jest pierwotne czy retransmitowane. Retransmitowane zamówienia są ignorowane.

2.2 Awaria serwera

Może dojść do dwu rodzajów awarii serwera:
- Awaria przed wykonaniem zamówienia
- Awaria po wykonaniem zamówienia

PDF created with pdfFactory Pro trial version www.pdffactory.com



Rys. 2-2 Secenariusze awarii wykonania polecenia w RPC

W obydwu przypadkach nie będzie wysłana odpowiedź. Klient nie potrafi odróżnić jaki jest typ awarii serwera. Możliwe są trzy rodzaje działania:

1. Ponawianie komunikatu zamawiającego przez klienta.
2. Filtrowanie powtórzonych zamówień przez serwer.
3. Retransmitowanie kopii zaginionych odpowiedzi przez serwer.

Prowadzi to do różnych semantyk RPC:

Semantyka wywołania ewentualnego (*ang. maybe*)
Realizacja:

- Nie stosuje się powtarzania komunikatu zamawiającego,
- Nie stosuje filtracji zamówień
- Nie stosuje powtarzania odpowiedzi.

Rezultat:

- Nie wiadomo czy procedura się wykonała.

Semantyka wywołania co najmniej jednokrotnego*(ang. at-least once)*
Realizacja:

- Stosuje się powtarzanie zamówień gdy odpowiedź nie nadejdzie w określonym czasie.
- Nie stosuje się filtracji powtórzeń.

Rezultat:

- Klient otrzyma w końcu wynik lub informację że serwer nie działa
- Brak informacji ile razy wykonała się procedura.

PDF created with pdfFactory Pro trial version www.pdffactory.com

Semantyka wywołania co najwyżej jednokrotnego *(ang. at-most once)*
Realizacja:

- Stosuje się powtarzanie zamówień klienta gdy odpowiedź nie nadejdzie w określonym czasie.
- Stosuje się filtrację powtórzonych zamówień.
- Stosuje się retransmisję zaginionych odpowiedzi

Rezultat:

- Klient otrzyma w końcu wynik lub informację że serwer nie działa
- Wiadomo że operacja wykonała się raz albo wcale

Ta semantyka jest preferowana w RPC.

Nazwa semantyki wywołania RPC	Ponowny komunikat zamawiający	Filtracja powtórzeń	Powtórne wykonanie procedury lub retransmisja odpowiedzi
Ewentualne	*Nie*	*Nie dotyczy*	*Nie dotyczy*
Co najmniej jednokrotne	*Tak*	*Nie*	*Powtórne wykonanie procedury*
Co najwyżej jednokrotne	*Tak*	*Tak*	*retransmisja odpowiedzi*

Semantyka wywołań RPC uwzględniająca awarie

Uwaga!
Możliwość awarii serwera istotnie odróżnia systemy zcentralizowane od rozproszonych.

PDF created with pdfFactory Pro trial version www.pdffactory.com

2.3 Awarie klienta

Gdy klient po wysłaniu ulegnie awarii po stronie serwera aktywny jest proces na który nikt nie czeka. Jest on nazywany procesem sierocym (*ang. orphan*). Jego istnienie jest szkodliwe gdyż zajmuje zasoby, blokuje pliki, semafory itd.

W pozbywaniu się sierot istnieją cztery rozwiązania:

Eksterminacja (ang. extermination)
Namiastka klienta zapisuje na trwałym nośniku informacje o wysłaniu zamówienia. Gdy (o ile) się wznowi przesyła do serwera zlecenie zakończenia (eksterminacji) poprzedniego zamówienia. Rozwiązanie to będzie nieskuteczne gdy:

- Klient się nie wznowi
- Nastąpi awaria sieci

Sieroty też mogą używać RPC tworząc dalsze sieroty – osierocone potomstwo (*ang. grandorphan*)

Reinkarnacja (*ang. reincarnation*)
Ponownie uruchomiony klient nadaje do wszystkich serwerów informację o nadejściu nowej epoki. Procesy należące do tego klienta z poprzedniej epoki mają być zakończone.

Łagodna reinkarnacja (*ang. gentle reincarnation*)
Ponownie uruchomiony klient nadaje do wszystkich serwerów informację o nadejściu nowej epoki. Serwer odnajduje procesy bez właściciela i je likwiduje.

Wygaśnięcie (ang. expiration)
Każda z rozpoczętych procedur otrzymuje na działanie kwant czasu T. Gdy go wyczerpie musi jawnie poprosić o następny. Nie wiadomo jaki ma być okres T.

PDF created with pdfFactory Pro trial version www.pdffactory.com

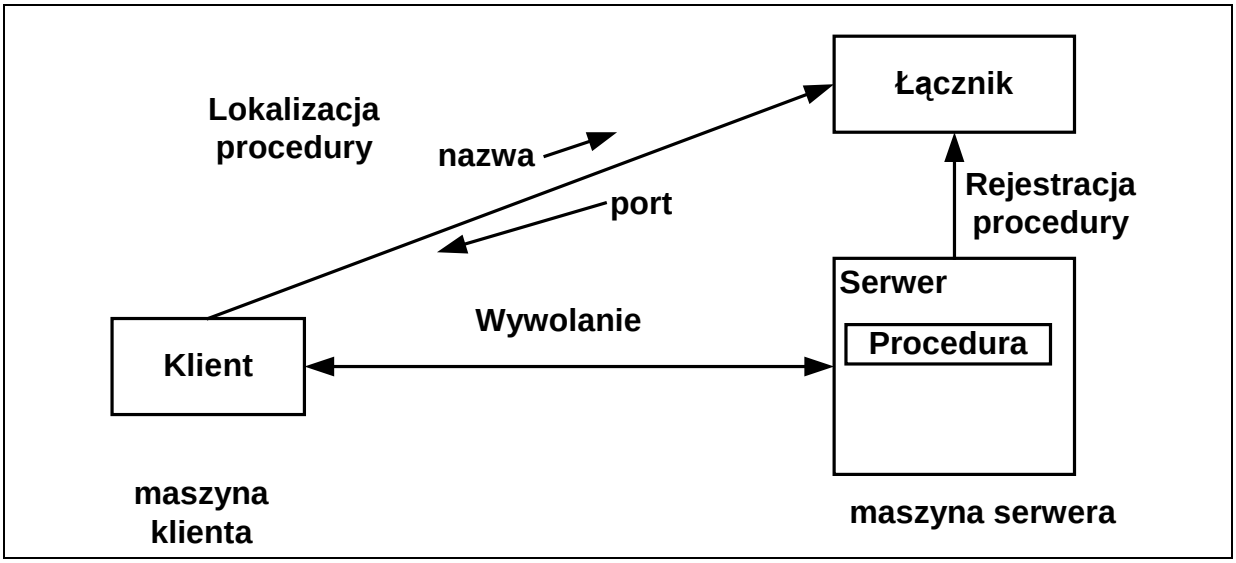
3. Wiązanie dynamiczne

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. 3-1 Rola łącznika w RPC

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

Funkcje interfejsowe łącznika

PDF created with pdfFactory Pro trial version www.pdffactory.com

Funkcje interfejsowe łącznika:

- **`void Rejestruj(NazwaUsługi, PortSerwera, Wersja)`**
- **`void Usuń(NazwaUsługi, Wersja)`**
- **`int  Szukaj(NazwaUsługi, Wersja)`**

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.

PDF created with pdfFactory Pro trial version www.pdffactory.com

Zagadnienia dotyczące implementacji

Implementując RPC należy dokonać wyboru protokołu;

1 Połączeniowy czy bezpołączeniowy
2 Standardowy (IP, UDP, TCP) czy specjalny

Ad 1

Zaletą protokołu połączeniowego jest niezawodność wadą spadek wydajności. W praktyce większość systemów RPC działa w sieciach lokalnych w których niezawodność jest i tak duża. Zatem częściej stosuje się protokoły bezpołączeniowe.

Ad 2

Wiele protokołów RPC używa jako podstawy protokołu IP co daje następujące korzyści:

1 Protokół jest zaprojektowany – oszczędność pracy
2 Protokół już zaimplementowany i przetestowany
3 Protokół IP obsługiwany we wszystkich systemach Uniksowych
4 Pakiety IP i UDP obsługiwane przez większość sieci.

Ujemną stroną jest wydajność. Pakiet IP ma 13 pól z których wykorzystywane są 3 pola – adres źródła, adres przeznaczenia i długość pakietu.

W RPC przekazuje się dużą ilość danych – korzystniejsza większa długość pakietu. W UDP ograniczenie długości pakietu do 8 KB.

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

PDF created with pdfFactory Pro trial version www.pdffactory.com

4. Standard XDR - przekazywanie parametrów

Maszyny mogą się różnić sposobem reprezentacji danych. Stąd konieczność konwersji do postaci kanonicznej. Jedną z metod rozwiązania problemu jest opracowany w firmie Sun system XDR (*ang. eXternal Data Reprezentation*).

Własności XDR:
- XDR umożliwia reprezentację danych w sposób niezależny od komputera.
- Jest to język opisu danych i narzędzie do ich konwersji.
- Nie jest zależny od żadnego szczególnego języka.

XDR stosuje konwencje:
- Liczby całkowite kodowane jako „big endian” – starsze bajty mają niższe adresy
- Liczby rzeczywiste kodowane w formacie IEEE
- Typy danych mają zawsze wielokrotność 4 bajtów (krótsze wypełniane zerami)
- Konwersja i dekodowanie prowadzone są zawsze
- Obowiązkiem nadawcy i odbiorcy jest znać typy przekazywanych danych

Korzystanie z XDR
Kodowanie i dekodowanie prowadzi się za pomocą funkcji XDR a dane przekazywane są za pośrednictwem potoków XDR.

Potok XDR – ciąg bajtów w którym dane reprezentowane są w postaci XDR. Są trzy typy potoków:
- standardowe wejście / wyjście,
- w pamięci,
- komunikaty.

Filtr XDR – procedura kodująca lub dekodująca określony typ danych (liczba całkowita, rzeczywista, znak, tablica).Filtr XDR piszą i czytają dane z potoku.

Rys. 4-1 Działanie mechanizmu XDR

PDF created with pdfFactory Pro trial version www.pdffactory.com

Plik **<rpc/xdr.h>** zawiera typ XDR i oznaczenia operacji.
XDR_ENCODE – kodowanie danych w potoku
XDR_DECODE – dekodowanie danych w potoku
XDR_FREE – zwalnianie pamięci

Potok w standardowym wejściu – wyjściu

```
xdrstdio_create(XDR *handle, FILE *file,
                enum xdr_op op)
```

handle – identyfikator potoku XDR
file - wskaźnik do otwartego pliku typy standardowe wejście/ wyjście
op - rodzaj operacji (XDR_ENCODE , XDR_DECODE)

PDF created with pdfFactory Pro trial version www.pdffactory.com

```
#include <stdio.h>
#include <rpc/rpc.h>
main()   {      // pisanie
   XDR xdrs;
   long i;
   xdrstdio_create(&xdrs, stdout, XDR_ENCODE);
   for (i = 0; i < 8; i++)  {
      xdr_long(&xdrs, &i))
   }
}
```

Program zapisu **writer**

```
#include <stdio.h>
#include <rpc/rpc.h>
main()  { // czytanie
   XDR xdrs;
   long i, j;
   xdrstdio_create(&xdrs, stdin, XDR_DECODE);
   for (j = 0; j < 8; j++) {
      xdr_long(&xdrs, &i);
      printf("%ld ", i);
   }
   printf("\n");
}
```

Program czytania **reader**

Użycie:

```
$ writer | reader
0 1 2 3 4 5 6 7
```

Potok w pamięci

```
xdrmem_create(XDR *handle, void *adr, int size,
                xdr_op op)
```

handle – identyfikator potoku XDR
adr - wskaźnik na bufor w pamięci
size - wielkość bufora
op - rodzaj operacji (XDR_ENCODE , XDR_DECODE)

Funkcja tworzy potok w pamięci do którego można pisać i czytać.

PDF created with pdfFactory Pro trial version www.pdffactory.com

Potok komunikatów

```
xdrrec_create(XDR *handle, int sendsize,
              int recsize,char *iohandle,
              int (*readproc)(),
              int (*writeproc)())
```

handle – identyfikator potoku XDR
sendsize, recsize - rozmiary bufora
iohandle – kanał komunikacji (gniazdko, strumien)
readproc – funkcja wykonywana gdy bufor pusty
writeproc – funkcja wykonywana gdy bufor pełny

Funkcja tworzy potok wysyłany na strumień wejścia wyjścia *FILE lub gniazdko.

Podział na komunikaty
Potok komunikatów umożliwia podział danych na komunikaty.

Zaznaczenie końca rekordu przy zapisie:
`xdrrec_endofrecord(XDR *handle, int sendnow)`

Odczyt rekordu (do znacznika końca rekordu):
`xdrrec_skiprecord(XDR *handle, int sendnow)`

Filtry XDR
- Filtry proste – umożliwiają konwersję typów prostych: char, int, long, float, double
- Filtry złożone - umożliwiają konwersję typów złożonych:

Filtry proste:
`xdr_int (XDR *xdrs, int *ip)` – konwersja typu int
`xdr_long (XDR *xdrs, int *ip)` – konwersja typu long int
`xdr_float (XDR *xdrs, float *ip)` – konwersja typu float
`xdr_double(XDR *xdrs, double *ip)` – konwersja typu double

PDF created with pdfFactory Pro trial version www.pdffactory.com

string	Łańcuch znaków
opaque	Tablica bajtów ustalonej wielkości
bytes	Tablica bajtów zmiennej wielkości
vector	Tablica danych dowolnego typu ustalonej wielkości
array	Tablica danych dowolnego typu zmiennej wielkości
union	Unia
reference	Wskaźnik
pointer	Wskaźnik wraz z rozpoznawaniem wskaźnika NULL, umożliwia obsługę list

Lista typów złożonych obsługiwanych przez XDR

PDF created with pdfFactory Pro trial version www.pdffactory.com

5. Tworzenie aplikacji RPC w Sun RPC

5.1 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

PDF created with pdfFactory Pro trial version www.pdffactory.com

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

PDF created with pdfFactory Pro trial version www.pdffactory.com

5.2 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:

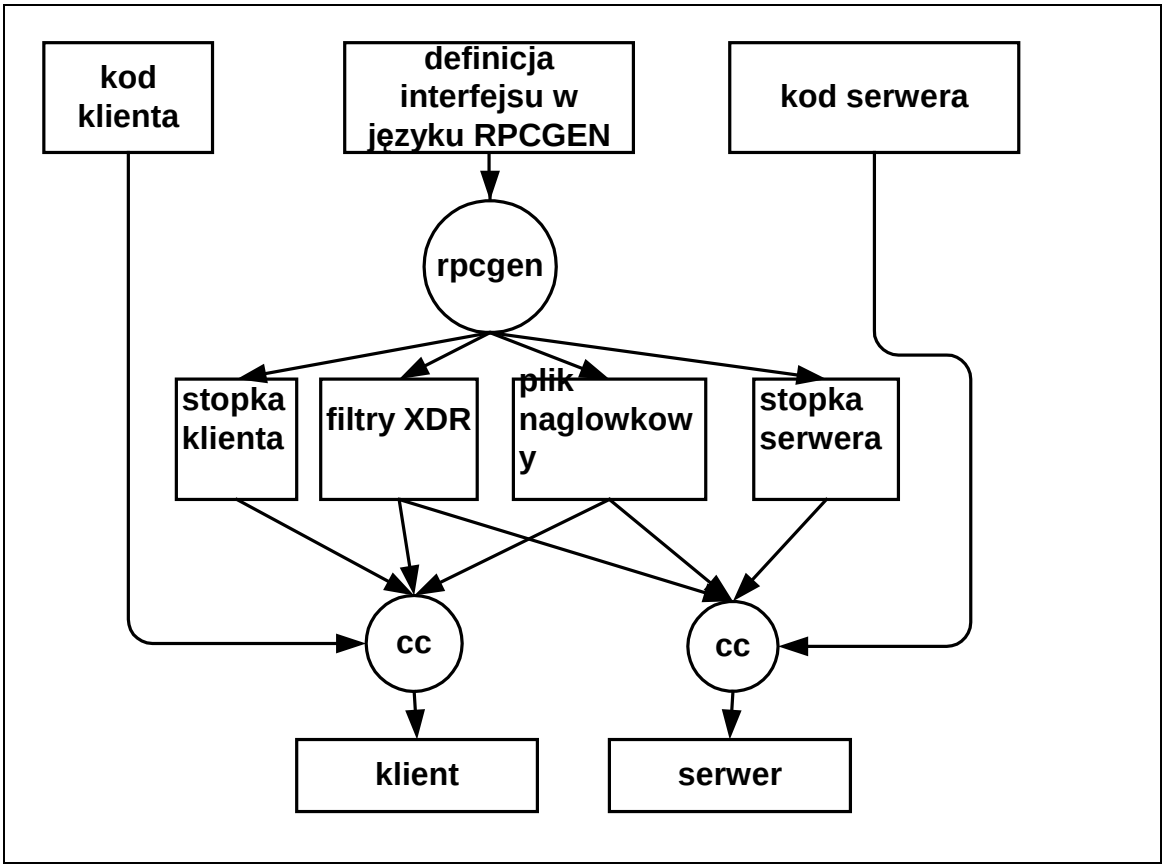
1. Wywołanie odległej procedury dopuszcza tylko jeden argument wywołania i zwraca jeden wynik.
2. Gdy występuje więcej elementów to należy umieścić je w strukturach.
3. W programach klienta i serwera argumentem lub wynikiem jest wtedy wskaźnik na strukturę.

PDF created with pdfFactory Pro trial version www.pdffactory.com

5.3 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. 5-1 Tworzenie aplikacji przy pomocy prekompilatora RPCGEN

PDF created with pdfFactory Pro trial version www.pdffactory.com

Przykład: procedura pisząca i czytająca komunikaty przekazywane pomiędzy maszynami

```
struct komunikat {
     int liczba;
     int numer;
};

program PROG1 {
    version VER1 {
         int pisz(komunikat) = 1;
         int czytaj(void) = 2;
    } = 1;
} = 0x30000003;
```

Interfejs **examp2.x** aplikacji przekazywania komunikatu w 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:

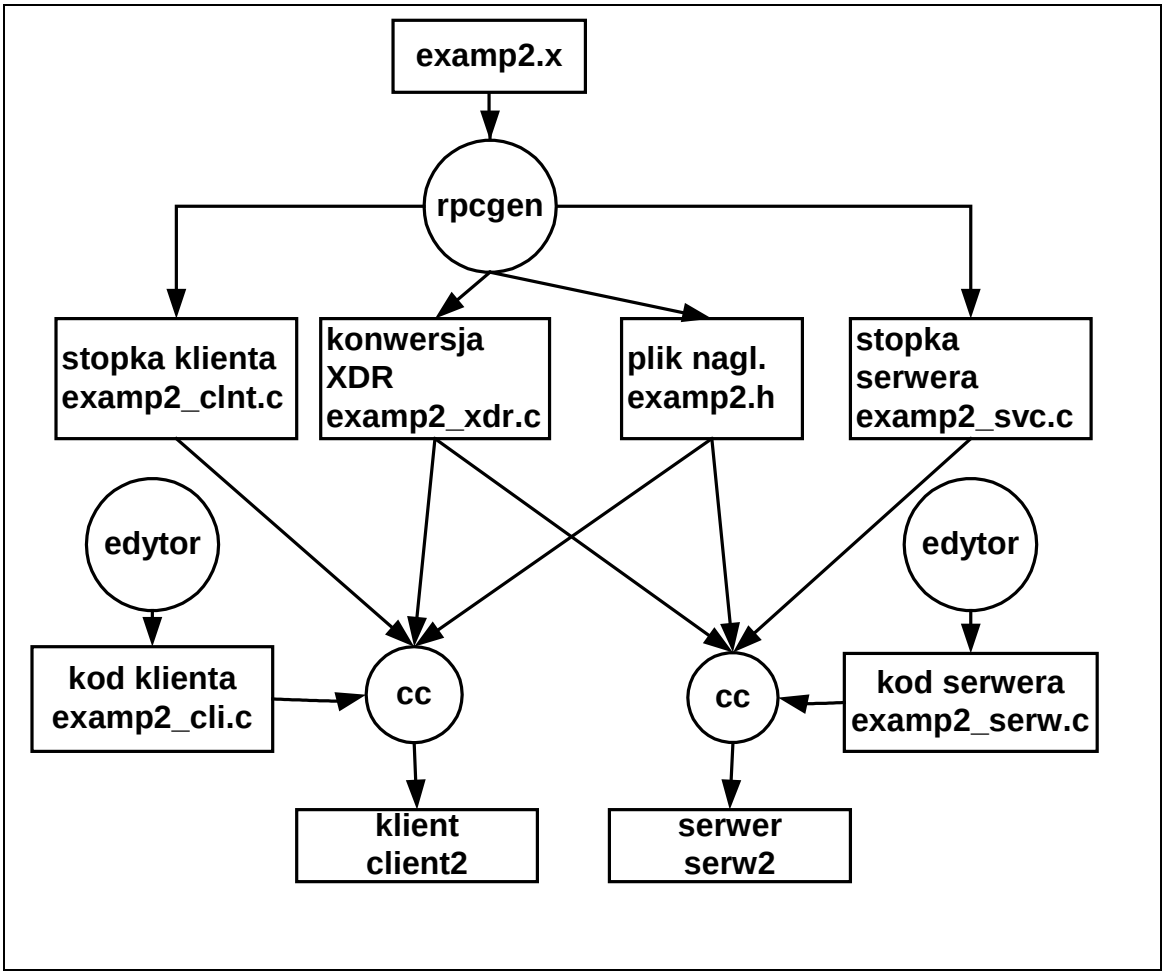
$ rpcgen examp2

Wynik:

examp2_clnt.c – Namiastka klienta (ang. client stub)
examp2_svc.c – Namiastka serwera (ang. server stub)
examp2_xdr.c - plik konwersji danych
examp2.h - plik nagłówkowy

PDF created with pdfFactory Pro trial version www.pdffactory.com

Należy dopisać:
examp2_serw.c – plik serwera
eamp2_cli.c – plik klienta



Rys. 5-2 Przykład tworzenia aplikacji RPC

PDF created with pdfFactory Pro trial version www.pdffactory.com

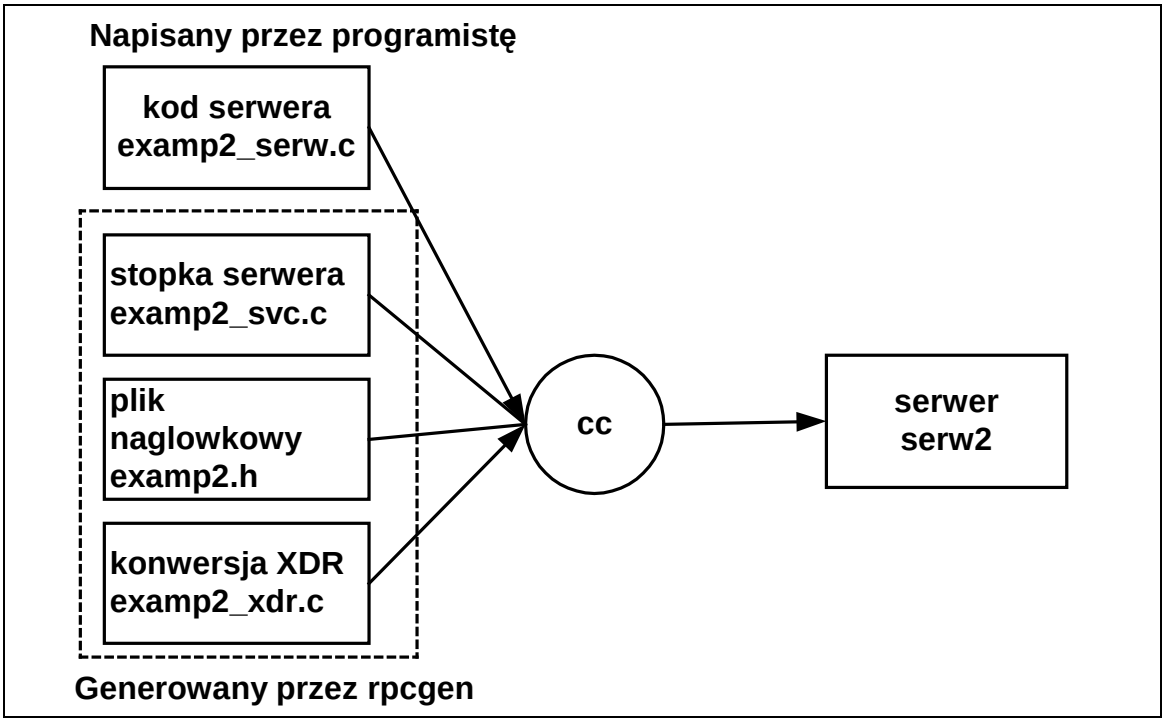
```
// Definicja danych
struct komunikat {
    int liczba;
    int numer;
};

// Definicje funkcji klienta
extern  int * pisz_1(komunikat *, CLIENT *);
extern  int * czytaj_1(void *, CLIENT *);

// Definicje funkcji serwera
extern int *pisz_1_svc(komunikat*, struct svc_req *);
extern int *czytaj_1_svc(void *, struct svc_req *);
```

Fragment pliku examp2.h

5.4 Tworzenie serwera

Aby utworzyć serwer należy:

1. Utworzyć plik z kodem serwera **`examp2_serw.c`**
2. Zaimplementować w nim funkcje serwera (**`pisz_1_svc, czytaj_1_svc)`** których prototypy podane są w pliku nagłówkowym.
3. Skompilować pliki i połączyć w plik wykonywalny serw2



Rys. 5-3 Przykład tworzenia serwera w RPC

PDF created with pdfFactory Pro trial version www.pdffactory.com

Rys. 5-4 Przykład tworzenia aplikacji serwera

```
// Kod programu serwera
// Kompilacja:   cc examp2_serw.c examp2_svc.c
// examp2_xdr.c -o serw2
#include "examp2.h"
static int bufor = 0;  // Zawartość bufora
static int bnum = 1;   // Liczba wywołań

int *pisz_1_svc(komunikat * kom, struct svc_req *s){
   static int wynik;
   bnum++;
   bufor = kom->liczba;
   wynik = 1;
   return(&wynik);
}

int *czytaj_1_svc(void * x, struct svc_req *s) {
   static int wynik;
   wynik = bufor;
   return &wynik;
}
```

Kod programu serwera

PDF created with pdfFactory Pro trial version www.pdffactory.com

5.5 Tworzenie klienta:

Aby utworzyć kod klienta należy znać:
1. Interfejs serwera – plik **`nazwa.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")
```

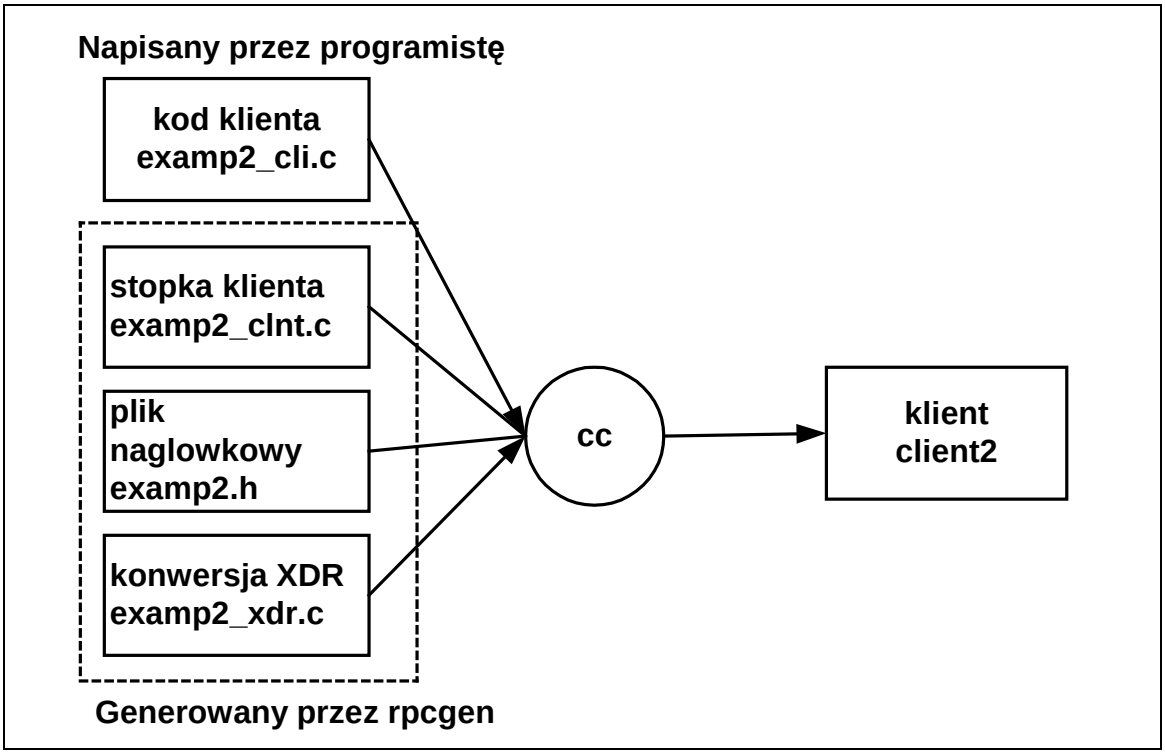
Wywołanie zdalnej procedury:

```
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`**.

PDF created with pdfFactory Pro trial version www.pdffactory.com



Rys. 5-5 Tworzenie klienta RPC

PDF created with pdfFactory Pro trial version www.pdffactory.com

```
// Kod programu klienta
// Kompilacja:   cc examp2_cli.c examp2_clnt.c
//               examp2_xdr.c -o client2
// Uruchomienie: zapis do serwera - client2 liczba
//               odczyt z serwera - client2
#include "examp2.h"
#include <stdio.h>
#include <stdlib.h>

#define HOST "192.168.1.222"

main(int argc, char * argv[]) {
    komunikat kom;
    int *wynik;
    int x;
    CLIENT *cli;
    cli = clnt_create(HOST, PROG1, VER1, "tcp");
    if (!cli) {
        clnt_pcreateerror(HOST);
        exit(1);
    }
    printf("Klient utworzony \n");

    if(argc > 1 ) {
      kom.liczba = atoi(argv[1]);
      wynik = pisz_1(&kom,cli);
      if(wynik != NULL) {
        printf("zapisano: %d\n",kom.liczba);
      } else {
        clnt_perror(cli, "zapis");
      }
   } else {
    wynik = czytaj_1(&x,cli);
    if(wynik != NULL) {
      printf("Odczytano:  %d \n",*wynik);
    } else {
        clnt_perror(cli, "odczyt");
    }
  }
}
```


PDF created with pdfFactory Pro trial version www.pdffactory.com

5.6 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`**.

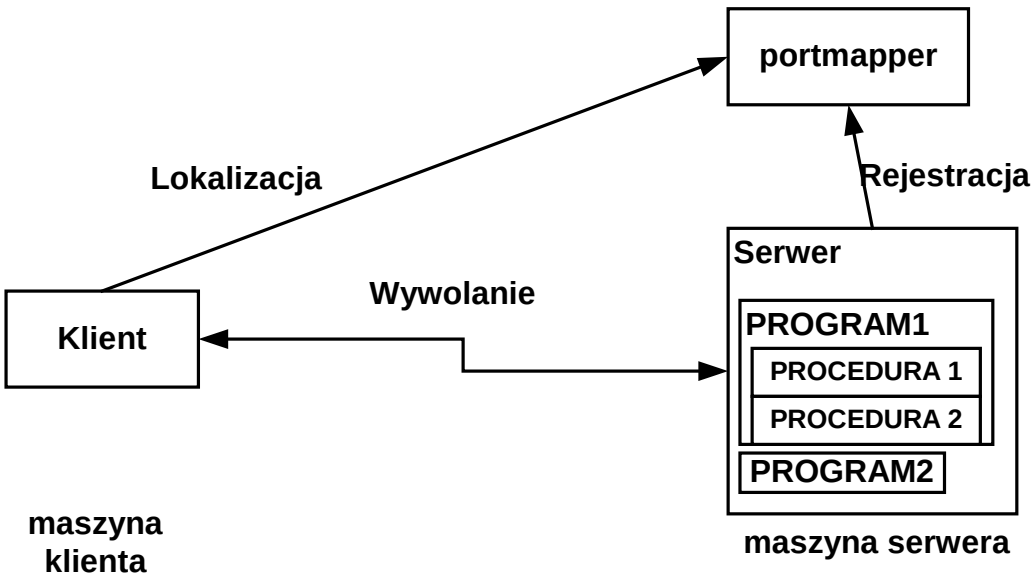
PDF created with pdfFactory Pro trial version www.pdffactory.com

```
$ rpcinfo –p nazwa_serwera

program      vers proto  port
    100000    2   tcp    111  portmapper
    100000    2   udp    111  portmapper
 805306372    1   udp   1020
 805306372    1   tcp   1022
 805306371    1   udp    935
 805306371    1   tcp    937
```

Uruchomienie aplikacji:

Na maszynie serwera (np. *zdalna*):
$ portmap & Uruchomienie łącznika (gdy nie jest uruchomiony)
$serw2 Uruchomienie serwera

Na maszynie klienta:
$ rpcinfo -p *zdalna* Testowanie łącznika
$ client2 Uruchomienie serwera



Rys. 5-6 Schemat rozproszonej aplikacji RPC

PDF created with pdfFactory Pro trial version www.pdffactory.com

6. Funkcje biblioteki RPC

Funkcje strony serwera

6.1 Tworzenie i obsługa identyfikatora transportu

Identyfikator transportu jest wskaźnikiem na strukturę typu SVCXPRT zdefiniowanej w pliku <rpc/rpc.h>

Utworzenie identyfikatora komunikacyjnego UDP:

```
SVCXPRT * svcudp_create(int sock)
```

```
sock – uchwyt gniazdka
```

Utworzenie identyfikatora komunikacyjnego TCP:

```
SVCXPRT *svctcp_create(int sock,int sendz,int
recvz)
```

sock – uchwyt gniazdka
sendz – wielkość bufora nadawczego
recvz – wielkość bufora odbiorczego

Kasowanie identyfikatora komunikacyjnego:

```
void svc_destroy(SVCXPRT *xprt)
```

xprt - identyfikator komunikacyjny

6.2 Rejestracja usługi RPC:

```
bool_t svc_register(SVCXPRT *xprt, int prognum,
int versnum, void(*dispatch)(), int protocol)
```

xprt – identyfikator komunikacyjny
prognum – numer programu
versnum – numer wersji
protocol – typ protokołu (IPPROTO_UDP lub IPPROTO_TCP)
dispatch – wskaźnik na procedurę wykonawczą

PDF created with pdfFactory Pro trial version www.pdffactory.com

Procedura wykonawcza (realizuje zlecenia):

```
void dispatch(struct svc_req *request, SVCXPRT
*xprt)
```

request – struktura zawierająca numer procedury, wersji i dane
xprt - identyfikator komunikacyjny

Wyrejestrowanie usługi RPC:

```
void svc_unregister (u_long prognum, int versnum)
```

prognum – numer programu
versnum – numer wersji

6.3 Dekodowanie argumentów procedury:

```
svc_getargs(SVCXPRT *xprt,xdrproc_t inproc, char
*in)
```

xprt - identyfikator komunikacyjny
inproc - funkcja dekodująca XDR
in - bufor w którym umieszczane są dane

Wysłanie odpowiedzi do klienta:

```
svc_sendreply(SVCXPRT *xprt, xdrproc_t outproc,
char *out)
```

xprt - identyfikator komunikacyjny
outproc - funkcja kodująca XDR
out - bufor w którym umieszczane są dane

Oczekiwanie na zlecenia klienta:

```
void svc_run (void);
```


PDF created with pdfFactory Pro trial version www.pdffactory.com

```
void main(){
 SVCXPRT *transp;
 ...
 transp = svcudp_create(sock);
 svc_register(transp, PROG1, VER1, prog1_1, proto)
 svc_run();
 exit(1);
}

prog1_1(struct svc_req *rqstp, SVCXPRT *transp) {
    union {
        komunikat pisz_1_arg;
    } argument;
    char *result;
    bool_t (*xdr_argument)(), (*xdr_result)();
    char *(*local)();

    switch (rqstp->rq_proc) {
    case NULLPROC:...return;
    case pisz:
        xdr_argument = xdr_komunikat;
        xdr_result = xdr_int;
        local = (char *(*)()) pisz_1_svc;
        break;


    case czytaj:
        xdr_argument = xdr_void;
        xdr_result = xdr_int;
        local = (char *(*)()) czytaj_1_svc;
        break;

    default: ...    return;
    }
    svc_getargs(transp, xdr_argument, &argument));
    result = (*local)(&argument, rqstp);
    svc_sendreply(transp, xdr_result, result))
    svc_freeargs(transp, xdr_argument,&argument));
    return;

}
```

Szkic serwera

PDF created with pdfFactory Pro trial version www.pdffactory.com

Procedura svc_run()

Procedura **`svc_run()`** powoduje zablokowanie procesu serwera na funkcji **`select(...)`** w oczekiwaniu na zlecenia klienta.
Kod źródłowy procedury **`svc_run()`** jest dostępny i można go modyfikować.

Okólniki

Możliwe jest wysyłanie rozgłoszenia (*ang. broadcast*) z wywołaniem procedury zdalnej. Metoda zapewnia wsparcie dla wysokiej dostępności.

```
int clnt_broadcast(prognum, versnum, procnum,
inproc, in,  outproc,  *out, eachresult)
```

`int`	**`prognum`**	Numer programu
`int`	**`versnum`**	Numer wersji
`int`	**`procnum`**	Numer procedury
`xdrproc_t`	**`inproc`**	Filtr wejściowy
`char *`	**`in`**	Bufor na argumenty
`xdrproc_t`	**`outproc`**	Filtr wyjściowy
`char *`	**`out`**	Bufor na wyniki
`bool_t`	**`(*eachresult)()`**	Funkcja wywoływana gdy pojawia się wyniki

Uwagi:

- Powtórzone odpowiedzi są ignorowane
- Transport tylko UDP
- Rozgłaszanie odbywa się do łącznika portmap. Stąd zdalne procedury powinny być tam zgłoszone.
- Maksymalna długość żądania 1400 bajtów, odpowiedzi 8800

PDF created with pdfFactory Pro trial version www.pdffactory.com

7. 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)

PDF created with pdfFactory Pro trial version www.pdffactory.com

8. 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.

PDF created with pdfFactory Pro trial version www.pdffactory.com