ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1996

Seria: INFORMATYKA z. 30 Nr kol. 1315

Andrzej KOWALCZYK
Arkadiusz TWARDOŃ
Krzysztof ZIÓŁKOWSKI

INTERFEJSY PROGRAMOWE, UMOŻLIWIAJĄCE KOMUNIKACJĘ MIĘDZY PROCESAMI W ŚRODOWISKU SIECIOWYM MICROSOFT NETWORK

Streszczenie. Artykuł omawia interfejsy programowe NetDDE, WinSock i RPC, pod kątem ich przydatności do tworzenia systemów o architekturze klient-serwer. Próba porównania dotyczy funkcjonalności, zakresu zastosowań i wydajności wybranych interfejsów.

PROGRAM INTERFACES FOR INTERPROCESS COMMUNICATION IN MICROSOFT NETWORK ENVIRONMENT

Summary. This article describes program interfaces NetDDE, WinSock and RPC, for how useful they are in creating client-server systems. The comparison concerns functionality, range of applications and performance of the choosen interfaces.

KOMMUNIKATION ZWISCHEN PROZESSES IN DER MICROSOFT WINDOWS - UMGEBUNG

Zusammenfassung. Der Beitrag stellt die Software-Schnittstellen NetDDE, WinSock und RPC aus Sicht ihrer Tauglichkeit für Systemerzeugung von Client-

Server-Architektur dar. Gewählte Schnittstellen werden auf Funktionalität, Anwendungsbereiche und Leistung verglichen.

1. Wstęp

Szybki rozwój systemów rozproszonych, a szczególnie oprogramowania działającego na podstawie modelu klient–serwer stawia przed projektantami nowe zadania. Istnieje potrzeba komunikowania się aplikacji nie tylko w obrębie sieci lokalnych ale także poprzez sieci rozległe (Internet). Tworzenie rozproszonych aplikacji wymaga stosowania odpowiednich narzędzi do komunikacji między częściami systemów. Podział zadań pomiędzy usługodawcę i usługobiorcę zmusza do zastanowienia się nad sposobem przekazywania poleceń i wyników operacji.

Do wyboru projektanci mają szeroki zestaw metod i interfejsów pozwalających na realizowanie komunikacji pomiędzy procesami w obrębie sieci komputerowej. Problem polega na wybraniu, odpowiedniego dla specyfiki działania projektowanego systemu, sposobu komunikacji.

Wyboru pomiędzy oferowanymi przez firmy softwarc'owe, różnymi koncepcjami komunikacji można dokonać kierując się różnymi kryteriami. W grę wchodzą kryteria ekonomiczne, łatwości instalacji i obsługi, wydajnościowe i inne. Z punktu widzenia projektanta systemu, dbającego o jak najlepszą pracę systemu korzystającego z coraz bardziej przeciążonych łącz komunikacyjnych, najważniejszy wydaje się wpływ wybranego mechanizmu komunikacji na przepustowość łącza danych.

Otwartość użytkowanych systemów, preferowana przez użytkowników końcowych, wymusza selekcję innego rodzaju. Projektanci systemów muszą się starać wybierać pomiędzy mechanizmami pozwalającymi na wykorzystanie w aplikacjach pracujących w środowisku różnych systemów operacyjnych i na różnych platformach sprzętowych. Otwartość projektowanego systemu nie może być bowiem ograniczona poprzez interfejs zastosowany do wymiany danych poprzez sieć.

Patrząc z tego punktu widzenia do badań wybrano trzy mechanizmy komunikacyjne, a mianowicie: NetDDE [1,2], RPC [1] i Winsock [1,3,4,5]. Porównano funkcjonalność i zakres zastosowań tych interfejsów, a przeprowadzone testy miały za zadanie określić, jaki wpływ na obciążenie zasobów systemu komputerowego i obciążenie sieci komputerowej ma każdy z badanych mechanizmów. Otrzymane wyniki pozwolić mają na określenie przydatności każdego ze sposobów komunikacji do realizacji różnego typu zadań.

2. Interfejs NetDDE

2.1. Dynamiczna wymiana danych w środowiskach Microsoft Windows

Dynamiczna wymiana danych (ang. Dynamic Data Exchange) jest jednym z podstawowych - obok schowka (ang. Clipboard) i OLE - mechanizmów komunikacji pomiędzy procesami w środowisku Windows 95 i Windows NT. Funkcje, komunikaty, typy danych i inne informacje niezbędne do tworzenia aplikacji DDE wchodzą w skład interfejsu programisty Windows (Win API i Win32 API). Bardzo wiele aplikacji Windows (np. MS Excel, MS Access, QuatroPro, AutoCAD...) potrafi udostępniać dane i korzystać z nich za pomocą DDE.

Poprzez DDE mogą komunikować się dwa procesy, z których jeden musi być serwerem DDE a drugi klientem DDE. Aplikacja, która inicjuje *konwersację* (ang. Conversation), jest klientem, aplikacja, która odpowiada na żądania klienta, jest serwerem.

Dostęp do danych w DDE jest zorganizowany w trójstopniową hierarchię. Każdy serwer DDE posiada *nazwę aplikacji* (ang. Application), jeden lub kilka zarejestrowanych *tematów* (ang. Topic Name), w obrębie każdego tematu może posiadać *zmienne* (ang. Item Name). *Zmienne DDE* są udostępniane dla klientów DDE w jednym z zarejestrowanych formatów: (np Text, BitMapa, ...). Zmienne DDE reprezentują pewien obszar pamięci operacyjnej, w której jest zapisana ich wartość. Pamięć ta może być współdzielona pomiędzy serwerem DDE i klientami.

Typowa konwersacja DDE składa się z następujących zdarzeń:

a) klient inicjuje konwersację, a serwer odpowiada;
b) aplikacje wymieniają dane na jeden z poniższych sposobów;
   - serwer wysyła dane na żądanie klienta (REQUEST);
   - klient wysyła dane do serwera (POKE);
   - klient żąda od serwera, aby wysyłał informacje o każdorazowej zmianie zmiennej DDE (ADVISE, warm data link);
   - klient żąda od serwera, aby wysyłał dane po ich każdorazowej zmianie (ADVISE, hot data link);
   - klient żąda od serwera wykonania określonego polecenia (EXECUTE);
c) klient lub serwer kończą konwersację.

Kiedy jest rozpoczęta konwersacja DDE, klient może ustanowić jedno lub kilka połączeń trwałych (ang. permanet data link). Takie połączenie jest mechanizmem, za pomocą którego serwer informuje klienta, kiedy wartość danej zmiennej DDE uległa zmianie. Połączenie trwałe jest utrzymywane aż do czasu, gdy przerwie je klient lub serwer. Rozróżniane są dwa rodzaje połączeń trwałych: ciepłe (ang. warm) i gorące (ang. hot). Połączenie ciepłe polega na

informowaniu klienta o zmianach danych ale bez przesyłania tych danych, klient musi zażądać danych w osobnej transakcji. Połączenie gorące polega na przesyłaniu danych do klienta każdorazowo po ich zmianie.

2.2. Zastosowanie DDE w środowisku sieciowym

Ponieważ DDE posługuje się komunikatami Windows i pamięcią współdzieloną do przekazywania danych między procesami, jest oczywiste, że ten interfejs może służyć tylko do komunikacji w obrębie jednego komputera. Ponieważ jednak taki sposób wymiany danych znalazł wśród twórców oprogramowania wielu zwolenników, to w związku z upowszechnieniem się architektury klient-serwer również w rozwiązaniach sieciowych zaistniała potrzeba umożliwienia komunikacji nie tylko w obrębie jednego komputera (pomiędzy procesami uruchomionymi na jednym komputerze) ale również w sieci. Interfejs NetDDE, zapewniający komunikację w sieciach komputerowych, został wprowadzony na rynek wraz z Windows NT i Windows for Workgroups.

NetDDE jest specjalną usługą w sieciach Microsoft Windows opartą na NetBIOS, która przekazuje komunikaty i dane pomiędzy procesami uruchomionymi na różnych komputerach w sieci. Usługa ta jest przeźroczysta z punktu widzenia aplikacji DDE, wymaga jednak pewnych zabiegów konfiguracyjnych dla poprawnego działania. Usługa ta nie jest domyślnie uruchamiana w standardowych instalacjach Windows. Dla komunikacji NetDDE niezbędne są *punkty wejściowe DDE* (ang. DDE share names). Punkt wejściowy DDE identyfikowany przez nazwę jest odpowiedzialny za komunikację z jednym serwerem DDE, może dotyczyć wszystkich tematów DDE danej aplikacji lub tylko jednego, wszystkich zmiennych DDE danego tematu lub tylko jednej zmiennej. Dla każdego punktu wejściowego można ustalić prawa dostępu lub/i hasło. Można zażądać uruchomienia odpowiedniego serwera DDE w przypadku próby komunikacji z danym punktem wejściowym. Do tworzenia punktów wejściowych, ich modyfikacji, uzyskiwania informacji o dostępnych punktach w sieci itp. służy specjalny zestaw funkcji *NetDDE API*.

2.3. NetDDE w środowisku Microsoft Windows for Workgroups

Uaktywnienie NetDDE następuje poprzez odpowiednie skonfigurowanie sekcji Network w Panelu Sterującym (ang. Control Panel) Windows. Punkty wejściowe NetDDE konfiguruje się edytując sekcję [DDEShares] w zbiorze SYSTEM.INI. Komunikacja NetDDE odbywa się za pomocą protokołu NetBEUI.

2.4. NetDDE w środowisku Microsoft Windows 95

Obsługą kumunikacji DDE z innymi komputerami w sieci zajmuje się specjalny proces. Jest to uruchamiany osobno 16-bitowy program NETDDE.EXE. Bez uruchomienia tego programu wszelkie odwołania do NetDDE kończą się niepowodzeniem, bez komunikatu o błędzie (tak jak by odpowiedni serwer DDE nie pracował). Konfiguracja punktów odpowiedzialnych za komunikacje NetDDE jest zawarta w *registry* [6,7] (*registry* jest to baza danych zawierająca informacje o konfiguracji systemów Windows NT i Windows 95). Interakcyjny program do ich tworzenia i modyfikacji nie wchodzi w skład standardowej instalacji Windows 95 ale jest dostępny za darmo w internecie (ftp.microsoft.com). Komunikacja NetDDE może odbywać się za pomocą dowolnego protokołu transportowego, obsługującego klienta pracującego w środowisku Microsoft Network.

2.5. NetDDE w środowisku Microsoft Windows NT 3.5x

Za komunikację NetDDE odpowiedzialny jest specjalny *proces usługowy* (ang. Service) konfigurowany w sekcji Services Panelu Sterującego. Kiedy ten proces usługowy nie pracuje, komunikacja NetDDE nie działa. Konfiguracja punktów wejściowych zawarta jest w *registry*. Program do interakcyjnego tworzenia i modyfikacji punktów wejściowych (DDESHARE.EXE) znajduje się w standardowej instalacji Windows NT.

Dostęp do punktów wejściowych jest chroniony standardowym mechanizmem ochrony dostępu (uprawnienia dla użytkowników). Normalne punkty wejściowe (ang. share) mogą być wykorzystywane przez serwery DDE pracujące jako systemowe procesy usługowe, aby komunikować się z serwerem pracującym jako aplikacja użytkownika punkt wejściowy musi być zaliczony do punktów uwierzytelnionych (ang. trusted share).

2.6. NetDDE w innych środowiskach operacyjnych

NetDDE jest produktem firmy Wonderware Corporation zakupionym przez Microsoft w 1992 roku. Oprócz wersji dla Microsoft Windows istnieją realizacje NetDDE w środowiskach UNIX i VMS oparte na NetBIOS, DECnet (DEC Pathworks) i Windows Sockets.

2.7. Zakres zastosowań

Interfejs DDE jest zaprojektowany w architekturze klient-serwer. Jest to interfejs wysokiego poziomu - użytkownik (programista) nie musi wiedzieć, w jaki sposób realizowana

jest komunikacja pomiędzy klientem i serwerem, ani też jakie medium jest w niej wykorzystywane. Usługi typu "hot link" i "warm link" upraszczają tworzenie systemów monitorujących zmiany wartości zmiennych DDE.

Interfejs DDE jest bardzo wygodny do tworzenia systemów konfigurowalnych przez użytkownika, ponieważ raz napisana aplikacja klienta lub serwera DDE może być używana do współpracy z innymi tego typu aplikacjami i wymaga to jedynie podania innych nazw tematów i zmiennych DDE.

2.8. Ocena wydajności

Pomiary wydajności interfejsu DDE wykonano w środowisku systemu Windows NT. Wybór tego systemu podyktowany był dostępnością narzędzi programowych umożliwiających takie pomiary. Uruchamiano aplikację testową i używano programu *Performance Monitor* do rejestrowania wskaźników wydajności.

Dwa podstawowe kryteria, jakimi kierowano się przy ocenie wydajności, to:

a) jak szybką transmisję można uzyskać za pomocą tego interfejsu ?

b) jak będą obciążane zasoby systemowe ?

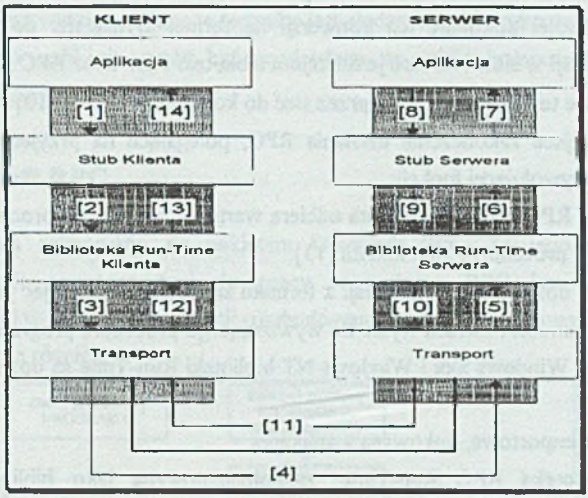
Testowy serwer DDE posiadał jeden temat DDE i trzy zmienne DDE, z których do celów pomiarów używano jednej. Uruchamiano od jednego do czterech procesów serwera, klient DDE pracował w środowisku Windows 95 i żądał transakcji *advise* typu "Hot Link". Serwer generował zmiany zmiennej DDE z największą możliwą szybkością w pętli. Planowano pomiary dla czterech rozmiarów zmiennej 16 bajtów, 256 bajtów, 4 kilobajty i 64 kilobajty. Pomiar dla 64kB okazał się niemożliwy do wykonania ze względu na zbyt małą wydajność systemu. Już dla zmiennej o rozmiarze 4kB występowało zjawisko znacznego opóźnienia w przychodących zgłoszeniach zmiany wartości zmiennej, przy znacznym obciążeniu zasobów systemu Windows NT. Przy małych długościach zmiennych DDE, z kolei, obciążenie było bardzo nieznaczne.

3. RPC

3.1. Zasada działania

RPC - ang.: Remote Procedure Call, czyli zdalne wywoływanie procedur, to narzędzie umożliwiające użytkownikom na pozornie bezpośrednie wywoływanie procedur, znajdujących się w zdalnym programie serwera. Zarówno aplikacja klienta, jak i serwera posiada w pamięci

swój obszar adresowy, czyli miejsce, gdzie może przechowywać dane, wykorzystywane podczas wywoływania zdalnych procedur. Rysunek 1 ilustruje zasadę wywoływania takich procedur z wykorzystaniem RPC:



Rys. 1. Zasada działania RPC
Fig. 1. The principle of RPC work

Jak to pokazano na ilustracji, aplikacja klienta zamiast wywoływać faktyczny kod, implementujący określoną procedurę, wywołuje procedurę typu *stub* [1]. Procedury te są kompilowane i linkowane z aplikacją klienta, jednak zamiast wykonywać faktyczny kod (implementujący zdalną procedurę), działają one w następujący sposób:

- Z obszaru pamięci klienta wydobywają odpowiednie parametry.
- Dokonują ich translacji na standardowy format reprezentacji danych sieciowych (ang.: *Standard Network Data Representation*), aby je przesłać za pomocą sieci.
- Aby móc wysłać do serwera parametry oraz odpowiednie żądanie, z biblioteki Run-Time klienta wywołują odpowiednie funkcje [2]. Funcje te, używając warstwy transportowej [3], przesyłają dane za pomocą sieci [4].

Z kolei na serwerze, po pobraniu danych z sieci [4], w celu wywołania procedury zdalnej, wykonywane są następujące kroki:

- Funkcje biblioteki RPC Run-Time serwera przyjmują żądanie [5] i wywołują procedurę *stub* serwera [6].
- Procedura ta wydobywa parametry z bufora sieciowego oraz dokonuje ich konwersji z formatu SNDR na format właściwy serwerowi.
- Następnie wywołuje na serwerze rzeczywistą procedurę [7].

Zdalna procedura jest teraz wykonywana, generuje (lub nie) parametry wyjściowe i wynik. Gdy zakończy działanie, w analogiczny sposób zwraca dane klientowi:

- Procedura zdalna zwraca dane do procedury *stub* serwera [8].
- Ta z kolei dokonuje ich konwersji na format wymagany do przeprowadzenia transmisji w sieci i zwraca je funkcjom biblioteki Run-Time RPC [9].
- Funkcje te transmitują dane przez sieć do komputera klienta [10] i [11].

Teraz ma miejsce zakończenie działania RPC, polegające na przyjęciu danych z sieci i zwrocie ich do wywołującej funkcji:

a) Biblioteka RPC Run-Time klienta odbiera wartości zwrotne od procedury zdalnej [12] i zwraca je procedurze *stub* klienta [13].
b) Ta z kolei dokonuje ich konwersji z formatu sieciowego, zapisując je w odpowiednim obszarze pamięci i zwraca wynik do wywołującego procedurę programu klienta [14].

Dla systemów Windows xxx i Windows NT biblioteki Run-Time są dostarczane w dwóch częściach:

a) biblioteka importowa, linkowana z aplikacją
b) oraz biblioteka RPC Run-Time, zaimplementowana jako biblioteka typu DLL (ang.: *Dynamic Link Library*).

Technika RPC jest swego rodzaju ewolucją modelu programowania proceduralnego. Dostarczając nowej kategorii specjalizowanych serwerów rozszerza ona model przetwarzania "*klient-serwer*".

Poprzez rozpraszanie przetwarzania w sieci RPC pozwala na znaczne zwiększenie możliwości obliczeniowych pojedynczego komputera, przerzucając część obliczeń na, zwykle bardzo szybkie, serwery.

Wykonanie procedury na podstawie RPC jest dla końcowego użytkownika praktycznie niewidoczne, gdyż całość jest obsługiwana poprzez procedury *stub* klienta i serwera oraz biblioteki Run-Time (patrz rysunek 1). Jest to więc bardzo dobre narzędzie, umożliwiające tworzenie aplikacji, realizujących przetwarzanie rozproszone.

3.2. Dlaczego RPC ?

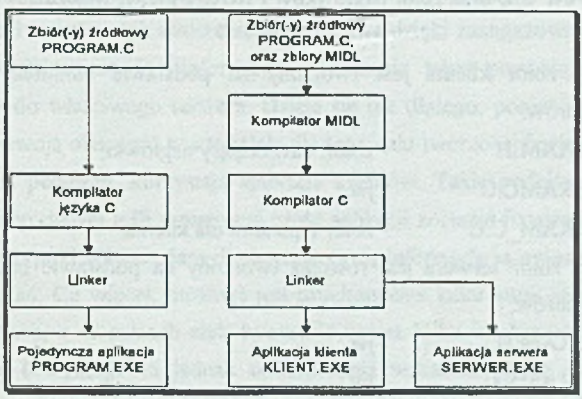
Podstawowe korzyści wynikające ze stosowania RPC są następujące:

- Ponieważ technika RPC jest stosunkowo prosta, przekształcanie zwykłych wywołań funkcji na wywołania procedur zdalnych nie jest skomplikowane, a dodatkowo oszczędza wiele czasu, potrzebnego zwykle na testowanie.
- Technika ta pozwala na odsunięcie użytkownika od interfejsu sieciowego; aby stosować RPC nie jest wymagana znajomość funkcji i protokołów sieciowych.

- RPC rozwiązuje występujący w sieciach heterogenicznych problem translacji danych. Stosując tę metodę pojedyncza aplikacja w ogóle nie musi się tym problemem zajmować.
- Podejście wykorzystujące tę technikę jest skalarne, tzn. wraz z rozprzestrzenianiem się sieci aplikacja może być rozdzielana na wiele komputerów jednocześnie pracujących w sieci.

3.3. Microsoft RPC

Podczas badań zajmowano się pakietem *Microsoft RPC*. Za jego pomocą, proces tworzenia aplikacji przystosowanej do pracy w technice RPC jest w stosunku do standardowego toku tworzenia aplikacji rozbudowany o dwa dodatkowe kroki. Poniższy rysunek pokazuje te różnice.



Rys. 2. Porównanie tworzenia aplikacji opartych na RPC i samodzielnych
Fig. 2. The comparison of creating RPC-based and alone-standing applications

Jak widać, proces tworzenia aplikacji realizującej przetwarzanie rozproszone, oparte na technice RPC zaraz na wstępie różni się od standardowego procesu przygotowywania samodzielnej aplikacji etapem kompilacji za pomocą kompilatora MIDL (ang.: *Microsoft Interface Definition Language*). Kompilator MIDL tworzy źródłowe zbiory "*.C" oraz zbiory typu "*stub*". Potem proces przebiega już podobnie do standardowego. Po kompilacji, przeprowadzonej przy użyciu kompilatora języka C, następuje linkowanie otrzymanych zbiorów "*.OBJ" z odpowiednimi bibliotekami, w wyniku czego powstają wykonywalne aplikacje klienta i serwera.

3.4. Tworzenie rozproszonej aplikacji

3.4.1. Niezbędne zbiory

W celu stworzenia sieciowej wersji przykładowego programu *"PROGRAM.EXE"* należy za pomocą pakietu *Microsoft RPC* stworzyć następujące zbiory:

- PROGRAMC.C - program klienta;
- PROGRAMS.C - program serwera;
- PROGRAMP.C - procedury zdalne;
- PROGRAM.IDL - zbiór, zawierający opis interfejsu identyfikacyjnego;
- PROGRAM.ACF - zbiór, zawierający atrybuty konfiguracyjne.

Kompilator MIDL, wykorzystując zbiory PROGRAM.IDL i PROGRAM.ACF, wygeneruje źródłowy zbior typu *stub* dla klienta: PROGRAM_C.C, źródłowy zbiór *stub* dla serwera PROGRAM_S.C oraz zbiór nagłówków PROGRAM.H, włączający zbiór nagłówków RPC.H.

Wykonywalny zbiór klienta jest tworzony na podstawie biblioteki Run-Time oraz następujących zbiorów:

- PROGRAM.H - zbiór, zawierający nagłówki;
- PROGRAMC.C - jw.;
- PROGRAM_C.C - zbiór typu *stub* dla klienta.

Wykonywalny zbiór serwera jest również tworzony na podstawie biblioteki Run-Time i następujących zbiorów:

- PROGRAM.H - jw.;
- PROGRAMS.C - jw.;
- PROGRAMP.C - jw.;
- PROGRAM_S.C - zbiór typu *stub* dla serwera.

3.4.2. Kolejność wykonywanych czynności

Podsumowując, można powiedzieć, że w celu stworzenia aplikacji przystosowanej do pracy w oparciu o technikę RPC (z wykorzystaniem *Microsoft RPC*), należy wykonać następujące kroki:

a) Stworzyć zbiór "*.IDL" (ang. *Interface Definition Language*), definiujący interfejs identyfikacyjny, typy danych oraz prototypy funkcji dla procedur zdalnych.
b) Stworzyć zbiór konfiguracyjny dla aplikacji.
c) Za pomocą kompilatora MIDL dokonać kompilacji zbioru "*.IDL". W wyniku zostaną wygenerowane zbiory typu *stub* (w języku C) oraz zbiór nagłówkowy dla klienta i serwera.

d) Do aplikacji klienta i serwera włączyć zbiór nagłówkowy, wygenerowany przez kompilator MIDL.

e) Stworzyć źródłowy program serwera, wywołujący funkcje RPC w celu udostępnienia klientowi informacji łączących, który następnie wywołuje funkcje **RpcServerListen**, nasłuchującą ewentualnych żądań od klientów. Stworzyć metodę pozwalającą na wykonanie Shut Down'u serwera.

f) Skonsolidować (ang. *links*) program klienta z właściwym zbiorem typu *stub* oraz bibliotekami Run-Time klienta.

g) Skonsolidować program serwera z właściwym zbiorem typu *stub*, procedurami zdalnymi oraz biblioteką RPC Run-Time serwera.

3.4.3. Funkcje Run-Time

Funkcje RPC są wykorzystywane w celu tworzenia logicznych połączeń pomiędzy aplikacjami klienta i serwera. Ich tworzenie jest możliwe dzięki zaangażowaniu kilku struktur danych, zawierających niezbędne informacje. Połączenia takie pozwalają na kierowanie zdalnych odwołań do właściwego serwera. Dzieje się tak dlatego, ponieważ każda aplikacja serwera rejestruje swoją obecność w specjalnie dla tego celu tworzonej bazie danych, z której, w celu zestawienia połączeń, korzystają aplikacje klientów. Takie podejście zapewnia dużą elestyczność systemu; nawet jeśli serwerowa część aplikacji zostanie fizycznie przeniesiona na inny komputer, to dopóki w bazie danych nie zmienią się informacje ją opisujące, klienci mogą dalej z niej korzystać. Co więcej, możliwe jest uruchomienie kilku kopii aplikacji serwera na różnych komputerach, a w ramach nich tworzenie nawet kilku implementacji dla tej samej procedury zdalnej (wymaga to jednak umieszczenia wskaźników do nowo tworzonych implementacji w specjalnych tablicach EPV - ang.. *Entry Poin Vector*.- mapujących odwołania do wywoływanych procedur zdalnych).

3.5. Podsumowanie

Podsumowując, należałoby stwierdzić, że mechanizm wykorzystujący RPC, oparty na architekturze klient-serwer, jest szczególnie przydatny do realizacji przetwarzania rozproszonego. Wiąże się to z faktem utrudnionego przesyłania za jego pomocą dużych ilości danych - w zasadzie przesyłane są jedynie parametry dla zdalnie wywoływanych procedur.

Tworzenie aplikacji jest bardzo wygodne - programista nie musi się troszczyć o to, jak dane będą przesyłane w sieci (kwestia protokołów i medium), ani o to, na którym komputerze zostanie uruchomiona aplikacja serwera.

Rozwiązanie jest również bardzo elastyczne - pozwala ono na definiowanie wielu wersji tej samej procedury zdalnej, bez konieczności wprowadzania zmian w aplikacjach klientów. Nie bez znaczenia jest również fakt, że jest to obecnie jeden z obowiązujących standardów.

4. Komunikacja pomiędzy procesami wg specyfikacji Microsoft Winsock 1.1

4.1. Historia rozwoju

Specyfikacja Microsoft Windows Sockets powstała z myślą o stworzeniu jednego interfejsu programisty (ang. *Aplication Programming Interface*), pozwalającego na tworzenie i rozwijanie oprogramowania wykorzystującego do komunikacji zasoby sieci komputerowej. Powstanie jednolitego zestawu funkcji realizujących komunikację pomiędzy procesami pozwala na tworzenie oprogramowania współpracującego z implementacjami niższych wastw modelu ISO/OSI, pochodzących od wielu producentów. Jest to możliwe dzięki zdefiniowaniu biblioteki funkcji możliwych do wywołania i związanej z nimi semantyki, która może być wykorzystana przez twórców oprogramowania. Ten sam zestaw funkcji musi być zaimplementowany przez twórców oprogramowania niższych warstw.

MS Windows Sockets, jako opis API, oparta jest na modelu tworzenia oprogramowania opracowanego na Uniwersytecie Kalifornijskim w Berkeley. Model Berkeley Sockets stał się standartem de facto, jeśli idzie o komunikację w sieciach opartych na protokole TCP/IP. Specyfikacja Windows Socket powstała na podstawie Berkeley Software Distribution Interprocess Communication (BSD IPC) w wersji 4.3.

4.2. Idea komunikacji pomiędzy gniazdami

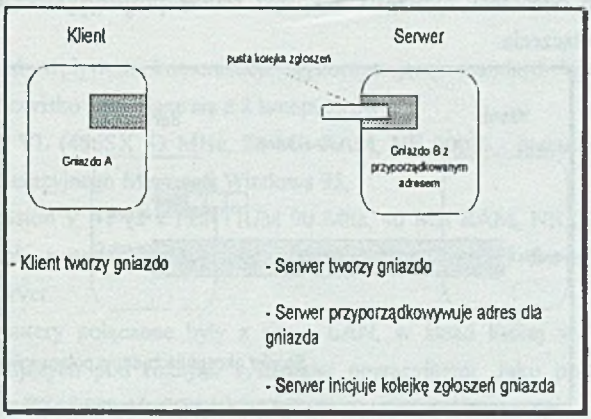
Standard BSD IPC, a co zatym idzie również Microsoft Winsock, pozwalają na tworzenie aplikacji wymieniających dane pomiędzy swoimi procesami. Co istotne, nie jest określone, czy procesy te muszą być uruchomione na tej samej maszynie, czy też komunikacja zachodzi przy wykorzystaniu sieci komputerowej. Standard pozwala na korzystanie z komunikowania się procesów bez ingerowania w to, w jaki sposób komunikacja pomiędzy nimi jest rzeczywiście realizowana. Cały mechanizm wywoływania wielu funkcji usługowych niższych warstw sieciowych ukryty jest pod pojęciem **gniazda komunikacyjnego** (ang. *socket*).

Gniazdo oraz **deskryptor gniazda** (ang. *socket decriptor*) i **przyporządkowanie adresu** (ang. *binding*) to podstawowe pojęcia pozwalające na zrozumienie zasady działania modelu BSD IPC.

a) Gniazdo - jest to punkt końcowy procesu komunikacji. Para gniazd połączonych ze sobą stanowi kanał komunikacyjny podobny do potoku.
b) Deskryptor gniazda - stanowi odpowiednik deskryptora pliku i jest używany jako argument funkcji używanych w modelu BSD IPC.
c) Przyporządkowanie adresu - jest konieczne, by gniazdo mogło być udostępnione innym procesom poprzez sieć komputerową.

Większość aplikacji modelu BSD składa się z dwóch osobnych części. Jedna z nich to proces żądający połączenia, czyli **klient**, druga to proces udostępniający i akceptujący połączenie czyli **serwer**.

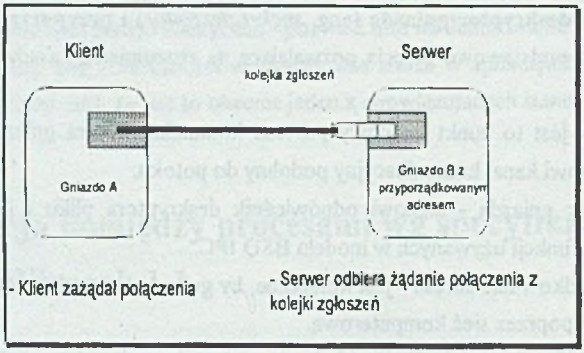
Proces nawiązywania połączenia przedstawiają poniższe rysunki.



Rys. 3. Stan aplikacji przed połączeniem gniazd
Fig. 3. The application state before socket connection

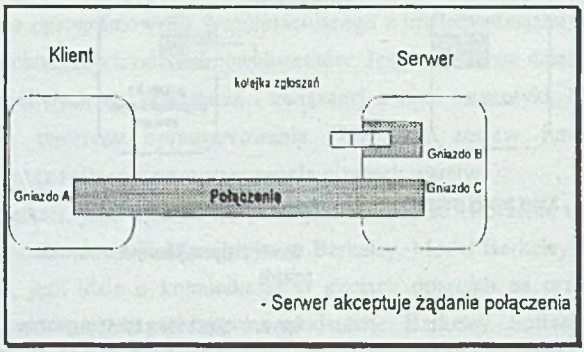
Serwer w momencie uruchomienia tworzy gniazdo i przyporządkowuje mu adres. Następnym krokiem jest stworzenie kolejki zgłoszeń dla nowo utworzonego gniazda. W kolejce tej gromadzone są żądania nawiązania połączenia.

Aplikacja klienta tworzy gniazdo i wysyła do serwera żądanie nawiązania połączenia.



Rys. 4. Sytuacja w momencie żądania połączenia
Fig. 4. The situation at the moment of connection request

Klient wysyła żądanie połączenia. Żądanie to pojawia się w kolejce zgłoszeń gniazda w serwerze. Po odebraniu zgłoszenia z kolejki serwer przystępuje do realizacji procesu nawiązywania połączenia.



Rys. 5. Sytuacja po nawiązaniu połączenia
Fig. 5. The situation after the connection has been established

Po zaakceptowaniu żądania połączenia realizowane jest połączenie pomiędzy gniazdem klienta oraz nowo utworzonym gniazdem serwera. Nowe gniazdo serwera posiada te same właściwości co gniazdo oryginalne. To ostatnie nadal kontynuuje nasłuchiwanie w oczekiwaniu na zgłoszenie kolejnego klienta.

Każde z gniazd używane w procesie komunikacji ma przyporządkowany własny typ oraz przynależy do procesu, który je utworzył. Gniazdo może istnieć tylko w obrębie jednej domeny komunikacyjnej.. Obecnie jedyną domeną, w obrębie której może występować komunikacja z wykorzystaniem Microsoft Winsock, jest domena Internetu, korzystająca z Internet Protocol Suite.

W chwili obecnej użytkownik ma do dyspozycji dwa typy gniazd:

a) gniazda strumieniowe zapewniające transmisję sekwencyjną, dwukierunkową, bez powtórzeń oraz z gwarancją dotarcia informacji do odbiorcy. Gniazda strumieniowe do komunikacji wykorzystują protokół TCP (ang. *Transmission Control Protocol*);
b) gniazda datagramowe - pozwalają na dwukierunkowy przepływ informacji. Nie gwarantują one jednak sekwencyjności oraz niepowtarzania przesyłanych danych. W wypadku gniazd datagramowych protokołem komunikacyjnym jest UDP (ang. *User Datagram Protocol*).

5. Opis stanowiska badawczego

5.1. Środowisko sprzętowe

Do badań nad wpływem komunikacji wykorzystującej standard Windows Sockets wykorzystano stanowisko składające się z 2 komputerów:

a) ALR Flyer VL (486SX 33 MHz, 20 MB RAM, NE 2000) - pracujący pod kontrolą systemu operacyjnego Microsoft Windows 95,
b) ALR Evolusion V ST (2 x PENTIUM 90 Mhz, 40 MB RAM, NE 2000) - pracujący pod kontrolą systemu operacyjnego Microsoft Windows NT Server 3.51 w wersji Domain Server.

Obydwa komputery połączone były z siecią LAN, w skład której wchodzi około 20 komputerów pracujących pod różnymi systemami operacyjnymi. Jako protokoły sieciowe wykorzystywane są IPX/SPX, NetBEUI i TCP/IP.

5.2. Narzędzia pomiarowe

Jako narzędzia mierzącego obciążenie zarówno sieci jak i samego komputera wykorzystywany był program Performance Monitor, wchodzący w skład "wyposażenia" systemu Microsoft Windows NT.

LITERATURA

[1] Microsoft® Win32 Software Development Kit, Windows™ NT 3.51 and Windows™ 95; 1985-1990 Microsoft® Corporation.

[2] Microsoft® Knowledge Base, Articles no. Q 125 703, Q 128 125, 1995 Microsoft® Corporation.

[3] HP 9000 Series 300/400 and 600/700/800 Computers Berkeley IPC Programmer's Guide, Hewlett Packard Edition 1, E0291.

[4] HP-UX Reference Manual, Hewlett Packard Third Edition E0892

[5] HP-UX HP 9000 Series 800, HP C Programmer's Guide, Hewlett Packard, Part No. 92434-90002.

[6] Microsoft® Windows 95 Resource Kit. 1995 Microsoft® Corporation.

[7] Microsoft® Windows NT 3.51 Resource Kit. 1995 Microsoft® Corporation.

Recenzent: Dr hab. inż. Adam Mrózek

Wpłynęło do Redakcji 18 grudnia 1995 r.

Abstract

This article describes three program interfaces: NetDDE, WinSock and RPC, used for interprocess communication.

Chapter one describes the data exchange problems, arising during process communication in different processing environments.

The first program interface NetDDE (*Network Dynamic Data Exchange*), is briefly described in chapter two. We have mentioned how NetDDE works in different network processing environments, like: Windows for Workgroups, Windows NT 3.5x, Windows 95 and UNIX systems. We have also reported the range of applications and functionality of this solution.

RPC (*Remote Procedure Call*) - the second tested interface - is described in the next chapter. There we have given the principles of RPC work, range of applications and the way of creating an application used for distributed processing, by means of Microsoft RPC.

Next interface - WinSock - was the main point of chapter four. We have described the history of its evolution, the idea of a "socket" and the principles of WinSock-based interprocess connections.

In the last chapter we have described the computers we used during our test.

All chapters contain our conclusions about described intefraces, their functionality and performance.