Politechnika Warszawska

Wydział Elektroniki i Technik Informacyjnych

Instytut Automatyki i Informatyki Stosowanej

Robert Kobyliński

Środowisko WDM do tworzenia aplikacji równoległych i rozproszonych na platformie MS Windows

v. 1.0

Warszawa 1999

Spis treści

1 Środowisko WDM do tworzenia aplikacji równoległych i rozproszonych na platformie MS Windows

Zamysłem przy projektowaniu tego systemu było stworzenie wirtualnej maszyny umożliwiającej prowadzenie obliczeń równoległych oraz symulacji przy wykorzystaniu wielu procesorów. Prowadzenie obliczeń równoległych na takiej maszynie jest możliwe poprzez dekompozycję podstawowego zadania na szereg mniej złożonych zadań obliczeniowych. Umiejętne przyporządkowanie zadań (najbardziej czasochłonne jednostkom o największej mocy obliczeniowej) pozwala na efektywne wykorzystanie zasobów systemu komputerowego jakim jest sieć stacji roboczych.

Biblioteka ***Windows Distributed Machine*** *(WDM)* nawiązuje w pewnym stopniu do systemów PVM, MPI oraz systemu symulacyjnego *CSA&S*. Mechanizmy tworzenia i niszczenia procesów są podobne do PVM i CSA&S, komunikacja przypomina tę z MPI. Zasadnicza różnica między systemami *WDM* i CSA&S polega na tym, że w systemie *CSA&S* obliczenia odbywały się przy wykorzystaniu jednego komputera, a *WDM* pozwala uruchamiać procesy na wielu komputerach pracujących w sieci. Ponadto *WDM* nie był projektowany jako system do symulacji złożonych systemów, choć oczywiście może być do tego celu wykorzystywany. Biblioteka ma zapewnić możliwość przeprowadzania symulacji bądź obliczeń na komputerach klasy PC pracujących na platformie Windows 95, Windows 98, Windows NT i Windows 2000. Użytkownik, czyli programista rozproszonych aplikacji numerycznych nie powinien przy tym wgłębiać się w szczegóły dotyczące komunikacji. Detale implementacyjne zaszyte są w procedurach biblioteki.

1.1 Budowa systemu

W systemie można wyróżnić cztery główne moduły oraz jednostki obliczeniowe:

Remote Execute Service (rexecsrv.exe, rexec95, rexeccmd.exe)

Zestaw programów (demonów) pozwalających zdalnie uruchamiać nasze procesy na odległych maszynach. Aktywacja demonów odbywa się ręcznie lub automatycznie jak to ma miejsce w przypadku serwisów Windows NT 4.0 i Windows 2000.

Windows Distributed Machine Console (WDMConsole - wdmcon.exe)

Program służący do definiowania projektu. Graficzny interfejs do komunikacji z jednostkami i jądrem systemu. Program pozwala programiście nadzorować obliczenia lub symulację. Moduł realizuje następujące funkcje:

- edycja struktury logicznej - przyporządkowanie procesów komputerom przez skojarzenie identyfikatora procesu i adresu IP komputera,
- zdale uruchomienie programów jednostek,
- zdalne usunięcie programów jednostek,

- wizualizacja stanu jednostek w trakcie obliczeń w postaci ikon,
- podglądanie wyników pośrednich (ekranów jednostek),
  - przekazywanie znaków wprowadzanych z klawiatury do programów jednostek pracujących na innym komputerze

Windows Distributed Machine Queue Server (WDMQS)

Jednostka centralna przechowująca i buforująca pakiety oraz informacje o aktualnym stanie jednostek. Jej zadaniem jest:

- kolejkowanie przesyłanych pakietów między jednostkami
- przesyłanie powiadomień o zmianie stanu jednostki do WDMConsole
- kierowanie zleceń wydanych przez WDMConsole (run, pause, stop, bye) do jednostki
- przechowywanie zawartości ekranów jednostek

Biblioteka ulib.lib (ulib.h)

Biblioteka napisana i skompilowana w Microsoft Visual C++ zawierająca funkcje do komunikacji, synchronizacji i zdalnego uruchamiania procesów. Przy pomocy zestawu procedur możliwe jest:

- zdalne uruchamianie procesów – programista będzie mógł uruchamiać swoje procesy na wybranych komputerach, korzystając z pracujących tam serwisów *Remote Execute Service,*
- identyfikacja jednostki – nawiązanie łączności, rejestracja, alokacja buforów, przydzielenie identyfikatora,
- wysyłanie i odbieranie komunikatów – przesłanie wiadomości (dowolnego ciągu bajtów) do innej jednostki działającej w systemie WDM,
- bariera – implementacja podstawowego mechanizmu synchronizacji procesów.

1.1.1 Remote Execute Service i Rexec95

System operacyjny Windows nie udostępnia mechanizmu, który pozwala na zdalne uruchamianie programów jak to ma miejsce w UNIX'ie *(rexec[1])***.** System realizuje to zadanie poprzez serwisy (demony), które muszą zostać uruchomione na każdym komputerze, który udostępnia swoje zasoby i pozwala na zdalne uruchamianie programów. System i funkcje biblioteczne komunikują się z serwisami i zlecają uruchomienie innego programu. W zależności od systemu operacyjnego zainstalowanego na komputerze, mamy do wyboru dwie aplikacje realizujące zadanie demona.

rexec95.exe – Windows 95, Windows 98, Windows NT 4.0, Windows 2000.

rexecsrv.exe – Windows NT 4.0, Windows 2000

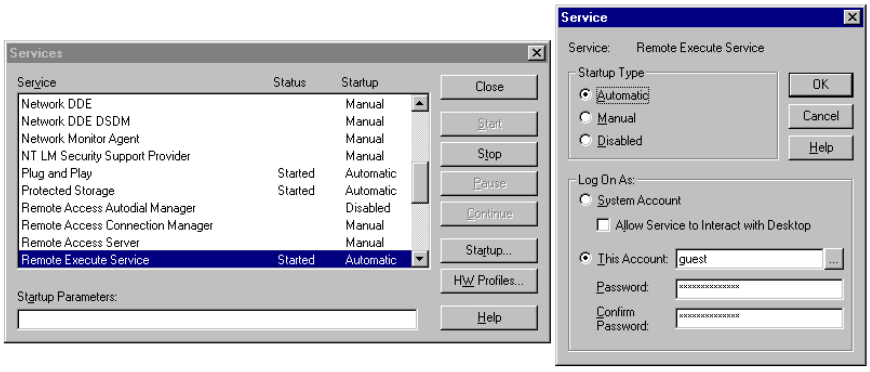
[1] *rexec - służy w systemie UNIX do uruchamiania programów na zdalnym komputerze, gdzie został uprzednio uruchomiony demon rexecd*

rexec95.exe

Program rexec95 to zwykła aplikacja uruchamiana przez użytkownika. Po uruchomieniu programu w prawym dolnym rogu pojawia się ikona informująca użytkownika o tym, że program jest aktywny. Zaznaczając ikonę i przyciskając prawy klawisz myszy możemy zakończyć pracę programu lub aktywować jego okno (tzn. na ekranie pojawi się okienko aplikacji rexec95.exe). Mamy wtedy dostęp do menu programu i możemy przejrzeć generowane w trakcie pracy komunikaty. Przeglądając wiadomości dowiemy się kto, kiedy i co uruchamia na naszym komputerze. Komunikaty generowane przez aplikacje są również pomocne w lokalizowaniu ewentualnych problemów w trakcie pracy systemu. Programy uruchamiane przez rexec95 pracują w kontekście zalogowanego użytkownika. Zasadniczą wadą takiego rozwiązania jest brak bezpieczeństwa i konieczność logowania i uruchomienia aplikacji w celu udostępnienia swojego komputera innym użytkownikom. Domyślnie rexec95 wykorzystuje do komunikacji port 1999. Jeżeli chcemy, aby rexec95 pracował na innym porcie to podajemy go jako argument wywołania programu.

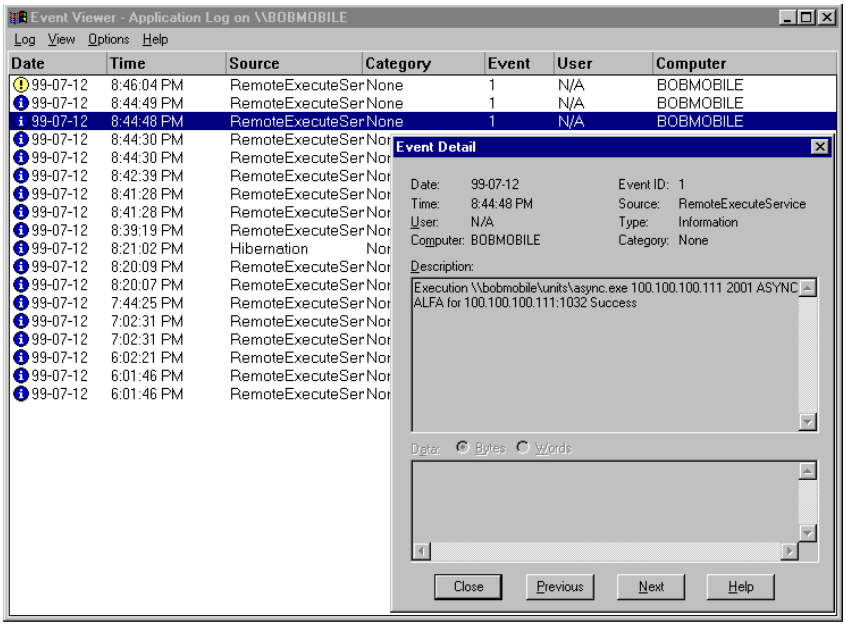
rexecsrv.exe

Z myślą o Windows NT oraz Windows 2000 został napisany serwis ***„Remote Execute Service”***. W skład programu wchodzą rexecsrv.exe i rexecsrv.dll. Serwis można skonfigurować tak, aby startował automatycznie po uruchomieniu systemu. Kolejna zaleta to możliwość zdefiniowania konta, na którym ma pracować serwis (Rys. 1-1). Wszystkie programy, które zostaną zdalnie uruchomione na naszym komputerze będą miały takie prawa do systemu i do sieci jak konto przypisane serwisowi. Podczas wyboru konta należy pamiętać o tym, że tylko konto systemowe *(System Account)* ma możliwość uruchomienia programu tak, aby jego wyniki pojawiły się na ekranie (w przypadku, gdy ktoś jest zalogowany). Jednak konto systemowe nie ma praw do sieci. To znaczy, że przed uruchomieniem programu należy go skopiować na dysk lokalny tak aby serwis nie musiał odwoływać się do zasobów udostępnianych w sieci. Jednocześnie tak uruchomiony program ma nieograniczony dostęp do naszego systemu. Dedykowane konto jest więc rozwiązaniem bezpieczniejszym.



Rys. 1-1 Konfiguracja serwisu Remote Execute Service

Podczas pracy serwis generuje informacje o wykonywanych operacjach, które przechowywane są w są w Dzienniku Systemu *(EventLogu)* i można je przejrzeć w dowolnej chwili. Na Rys. 1-2 widzimy, że 99-07-12 o godzinie 20:44 *Remote Execute Service* uruchomił na naszym komputerze program \\bobmobile\units\async.exe na życzenie użytkownika pracującego w sieci pod adresem 100.100.100.111. W *EventLogu* możemy także znaleźć wszelkie informacje o błędach, które wystąpiły podczas pracy serwisu.



Rys. 1-2 EventLog - komunikaty z Remote Execute Service

Przykładowy scenariusz może wyglądać następująco: Program jednostki *(u.exe)* znajduje się fizycznie na naszym komputerze *ALFA*. Chcemy go uruchomić na komputerze *BETA*, gdzie jest zainstalowany serwis pracujący na koncie *GUEST*. W pierwszej kolejności musimy stworzyć zasób np. "*pub*" na komputerze *ALFA* i nadać uprawnienia użytkownikowi komputera *BETA (BETA\guest)* do naszego zasobu *"pub*". Następnie umieszczamy tam kod programu jednostki *(u.exe)*. W takim przypadku jako argument programu rexec możemy podać nazwę unc[2] \\ALFA\pub\u.exe.

Oprócz możliwości zdalnego uruchamiania programów serwisy pozwalają także na ich zabicie. Serwis uruchamia program za pomocą funkcji API *CreateProcess()* i zwraca klientowi identyfikator stworzonego procesu. Klient może później wykorzystać ten identyfikator do zabicia procesu. Proces jest zabijany przy użyciu funkcji *TerminateProcess().*

Argumenty funkcji CreateProcess i TerminateProcess:

```
BOOL CreateProcess(
   LPCTSTR lpApplicationName,  // name of executable module
   LPTSTR lpCommandLine,       // command line string
   LPSECURITY_ATTRIBUTES lpProcessAttributes,  // process security attributes
   LPSECURITY_ATTRIBUTES lpThreadAttributes,   // thread security    attributes
   BOOL bInheritHandles,  // handle inheritance flag
```

[2] *UNC Universal Naming Convention - konwencja nazewnictwa komputerów i zasobów przyjęta w sieciach Microsoft Network* \\<server name>\<share point name>

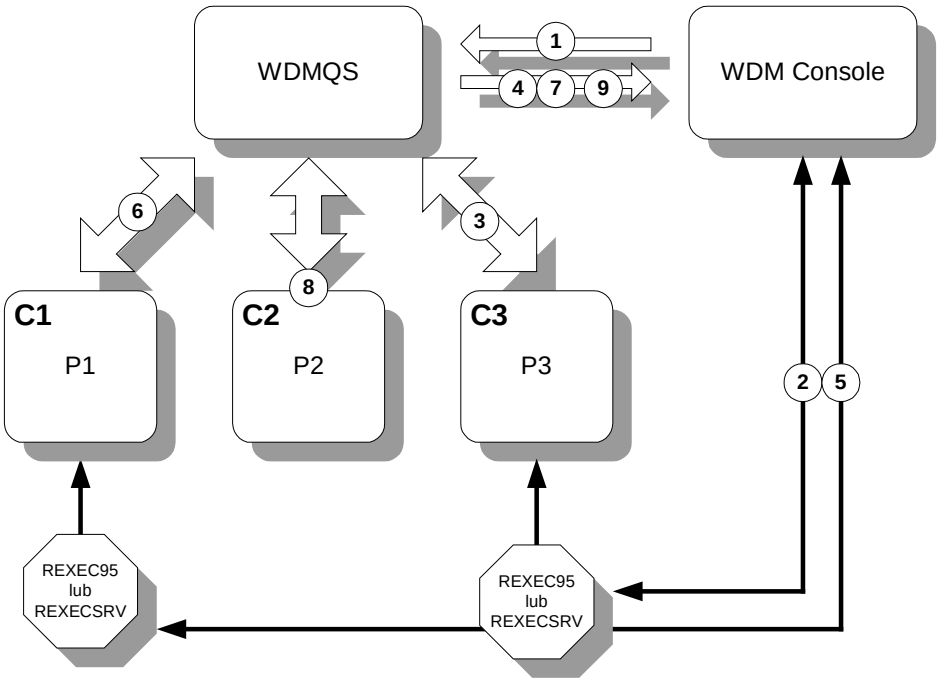
```
   DWORD dwCreationFlags, // creation flags
   LPVOID lpEnvironment,  // pointer to new environment block
   LPCTSTR lpCurrentDirectory,   // pointer to current directory  name
   LPSTARTUPINFO lpStartupInfo,  // pointer to STARTUPINFO
   LPPROCESS_INFORMATION lpProcessInformation  // pointer to PROCESS_INFORMATION
);
BOOL TerminateProcess(
   HANDLE hProcess, // handle to the process
   UINT uExitCode   // exit code for the process
);
```

Procedury do komunikacji z serwisami ***(wdm_rexec()*** *i* ***wdm_kill())*** znalazły się również w bibliotece WDM. Użytkownik może je wykorzystać do automatycznego uruchamiania programów jednostek obliczeniowych.

```
int  wdm_kill( char *host, WORD port,
               DWORD PID,
               char *username, char *password);
int wdm_rexec( char *host, WORD port,
               char *cmdline,
               char *username, char *password);
```

1.1.2 Komunikacja miedzy procesami obliczeniowymi i modułami systemu

Komunikacja między modułami odbywa się poprzez przesyłanie komunikatów. Warstwa transportowa oparta na TCP/IP, pozwala uruchamiać moduły i jednostki obliczeniowe na dowolnym komputerze pracującym w Internecie. TCP zapewnia niezawodność przesyłania pakietów między komponentami systemu. Sposób komunikacji między elementami systemu przedstawia Rys. 1-3



Rys. 1-3 Komunikacja między jednostkami

W pierwszej kolejności należy uruchomić WDMQS i WDM Console.

1. WDMConsole rejestruje projekt w WDMQS (Windows Distributed Machine Queue Server).	
2. WDMConsole komunikuje się z rexecsrv lub rexec95 na komputerze C3 i przekazuje informacje o jednostce, która ma zostać uruchomiona na danym komputerze. Serwis rexec uruchamia program jednostki P3 i przekazuje mu informacje na temat WDMQS (ip, port) oraz informacje o projekcie (project name, unit name).	
3. Jednostka P3 rejestruje się w WDMQS. Podając project name, unit name otrzymuje UID (Unit ID)	
4. WDMQS wysyła informacje do WDMConsole potwierdzając rejestrację jednostki P3 w systemie.	 P3
5. WDMConsole komunikuje się z rexecsrv lub rexec95 na komputerze C1 i przekazuje informacje o jednostce, która ma zostać uruchomiona na danym komputerze. Serwis rexec uruchamia program jednostki P1 i przekazuje mu informacje na temat WDMQS (ip, port) oraz informacje o projekcie(project name, unit name).	
6. Jednostka P1 rejestruje się w WDMQS. Podając project name, unit name otrzymuje UID (Unit ID).	
7. WDMQS wysyła informacje do WDMConsole potwierdzając rejestrację jednostki P1.	 P1
8. Jednostka P2 zostaje ręcznie uruchomiona na komputerze C2. Informacje na temat WDMQS (ip, port) oraz informacje o projekcie (project name, unit name) są podane jako parametry wywołania programu. Jednostka rejestruje się w WDMQS. . Podając projekt name, unit name otrzymuje UID (Unit ID).	
9. WDMQS wysyła informacje do WDMConsole potwierdzając rejestrację jednostki P2.	 P2

W trakcie obliczeń komunikacja między jednostkami odbywa się również za pośrednictwem jądra WDMQS. Jeżeli jednostka P1 ma wysłać komunikat do jednostki P3 wykonywane są następujące kroki:

1. P1 wywołuje funkcje wdm_send(P3…)
2. Pakiet zawierający nagłówek oraz treść komunikatu dociera do WDMQS
3. WDMQS umieszcza pakiet w buforze przypisanym jednostce P3
4. Kiedy jednostka P3 wywoła funkcje wdm_recv(P1…) otrzyma od WDMQS pakiet wysłany przez jednostkę P1

Stosując terminologię z MPI ***wdm_send()*** przesyła komunikat w sposób asynchroniczny, nieblokujący, a ***wdm_recv()*** asynchronicznie, blokująco.

Uwaga: *Należy pamiętać o tym, że WDMQS kolejkuje nadchodzące pakiety. Jeżeli jednostka wywoła funkcje* ***wdm_recv_any()*** *dostanie pakiet, który jest pierwszy na liście. Jeżeli jednostka wywoła funkcję* ***wdm_recv(xUID,…)*** *otrzyma pierwszy z pakietów jakie nadeszły od jednostki xUID. Wywołanie* ***recv*** *może być blokujące lub nie. Jeżeli w buforze*

jednostki znajdują się pakiety kontrolne (stop, pause wysłane na polecenie WDMConsole) to zostaną one przekazane w pierwszej kolejności, niezależnie od żądania jednostki.

1.1.3 Program WDMQS

Można powiedzieć, że *WDMQS (Windows Distributed Machine Queue Server)* jest sercem systemu. Program może pracować jako zwykła aplikacja lub jako serwis NT. Tu są kolejkowane wszystkie pakiety. Jądro *WDMQS* odpowiada także za alokację buforów jednostek. Do komunikacji z jednostkami *WDMQS* rezerwuje domyślnie port 2001. Można to zmienić podając inny numer portu jako parametr wywołania programu. *WDMQS* przechowuje wszystkie informacje o jednostkach (aktualny stan, zawartość ekranu, kolejkowane pakiety). W chwili pierwszego połączenia *WDMConsole* z *WDMQS* odbywa się rejestracja projektu. Podczas rejestracji *WDMConsole* przekazuje wszystkie dane o jednostkach, *WDMQS* alokuje bufory i inicjuje zmienne dotyczące projektu. W trakcie pracy *WDMQS* gromadzi dane napływające z jednostek. Takie rozwiązanie pozwala na odłączenie *WDMConsole* i ponowne połączenie z *WDMQS* np. z innego komputera. Jeżeli WDMConsole połączy się powtórnie zostanie on powiadomiony, że projekt jest już załadowany i aktywny. Wtedy *WDMQS* prześle aktualne dane o stanie jednostek do *WDMConsole*. Możliwe jest uruchomienie kilku instancji *WDMConsole* na różnych komputerach w tej samej chwili. Wszystkie będą jednocześnie pokazywać stan jednostek, ekrany lub przekazywać klawisze. Działanie *WDMQS* jest oparte na opisanej wcześniej funkcji ***select()*** z biblioteki gniazdek. Program prowadzi rejestr aktywnych połączeń z jednostkami i konsolami. W trakcie pracy czeka blokująco aż na jednym z obserwowanych gniazdek pojawią się dane. Może to być pakiet wysłany przez jedną jednostkę do drugiej, komunikaty sterujące z konsoli lub nowe informacje o zmianie zawartości ekranu jednostki. W nagłówku każdego komunikatu znajduje się jego identyfikator. W sumie rozróżnia się około 50 różnych wiadomości. Po odebraniu komunikatu *WDMQS* podejmuje właściwą akcję. Komunikaty obsługiwane są sekwencyjnie.

1.1.4 Konsola operatora - WDMConsole

Z tym modułem będziemy mieli najwięcej styczności podczas pracy z systemem. Program służy do zdefiniowania jednostek (nazwa, ID, host, linia poleceń *(command line)*) oraz zdalnego uruchomienia jednostek. W trakcie obliczeń na ekranie konsoli możemy obserwować dynamicznie zmieniający się stan jednostek. Każdy stan jest reprezentowany przez odpowiednią ikonę. Konsola pozwala także podglądać wyniki pośrednie (ekran) jednostek oraz interaktywnie wprowadzać dane z klawiatury do odległych jednostek. Takich możliwości nie dostarczają wspomniane pakiety PVM i MPI.

Stan jednostki na ekranie WDMConsole


- definiowanie jednostek; projekt nie jest zarejestrowany


- projekt zarejestrowany, jednostka nie uruchomiona


- do jednostki wysłano z konsoli polecenie *start*


- nieudana próba uruchomienia jednostki, błąd komunikacji z serwisem *Remote Execute Service*

 - jednostka została uruchomiona, ale nie zarejestrowała się w WDMQS

 - jednostka aktywna *(wywołano wdm_UnitRegister)*

 - do jednostki wysłano z konsoli polecenie *pause*

 - jednostka wywołała funkcję *wdm_barrier()*

 - jednostka zawieszona

 - do jednostki wysłano z konsoli polecenie *stop*

 - jednostka zakończyła obliczenia *(wdm_UnitStop)*

 - jednostka oczekuje na wprowadzenie danych *(wdm_scanf)*

 - jednostka oczekuje blokująco na dane od innej jednostki *(wdm_recv)*

1.2 Windows Distributed Machine Library - funkcje biblioteki ulib.lib

Do komunikacji między jednostkami oraz ich synchronizacji wykorzystujemy funkcje biblioteczne. Zgodnie z założeniami, warstwa sieciowa oparta na TCP/IP jest niewidzialna dla użytkownika. Pozwala to programiście operować na wyższym poziomie i skupić swoją uwagę na rozwiązywanym problemie. Korzystanie z biblioteki eliminuje konieczność kodowania warstwy komunikacyjnej, co niezaprzeczalnie skraca czas pisania aplikacji rozproszonej. Deklaracje funkcji dostępnych w bibliotece zawarte są w pliku ulib.h:

```
// Windows Distributed Machine library ulib.h
#include <windows.h>

//Funkcje Rejestrujące
WORD wdm_UnitRegister(int argc, char **argv);
void wdm_UnitUnregister();
void wdm_UnitStop();

// funkcje komunikacyjne
int  wdm_recv(int src , char  *ptr);
int  wdm_recv_any(WORD *rsrc, char *ptr);
int  wdm_async_recv(int src , char  *ptr);
int  wdm_async_recv_any(WORD *rsrc, char *ptr);
int  wdm_send(int dst , char  *ptr, int len);
int  wdm_send_over(int dst , char  *ptr, int len);
int  wdm_broadcast(char  *ptr, int len);
int  wdm_broadcast_over(char  *ptr, int len);
void wdm_barrier(int nUID, char *name, ...);
```

```
void wdm_barrier_all(char *name);
```

// funkcje wejścia wyjścia

```
void wdm_printf(char *fmt, ...);
void wdm_scanf(char *s);
```

// funkcje dodatkowe

```
int  wdm_kill( char *host, WORD port,
               DWORD PID, char *username, char *password);
int wdm_rexec( char *host, WORD port,
               char *cmdline, char *username, char *password);
void wdm_ParseArgs( int argc, char **argv, char *ip,
                    WORD *port, char *project, char *name);
```

Funkcje rejestrujące są niezbędne do prawidłowego działania jednostki. Jednostka musi się zarejestrować w WDMQS i wyrejestrować po zakończeniu obliczeń. Właściwą część aplikacji piszemy przy wykorzystaniu funkcji komunikacyjnych. Tak naprawdę, większość zadań możemy rozwiązać przy użyciu kilku z tych funkcji ***(wdm_barrier(), wdm_recv(), wdm_recv_any(), wdm_send())***. Jednak w szczególnych przypadkach, może zaistnieć potrzeba użycia funkcji dodatkowych. Rozszerzają one znacznie możliwości systemu, czyniąc go bardziej elastycznym i uniwersalnym.

Dokładny opis procedur bibliotecznych przedstawimy razem z przykładowymi programami, które będą z nich korzystać w rozdziale 2. Powinno to dopomóc czytelnikowi w tworzeniu własnych aplikacji. W tym rozdziale omówimy dokładniej to, co jest jedną z najtrudniejszych rzeczy w obliczeniach rozproszonych, a mianowicie metody synchronizacji procesów.

1.2.1 Synchronizacja procesów w systemie WDM

Do synchronizacji procesów w systemie WDM służy funkcja *wdm_barrier*.

void wdm_barrier(int nUID, char *name, ...);

gdzie

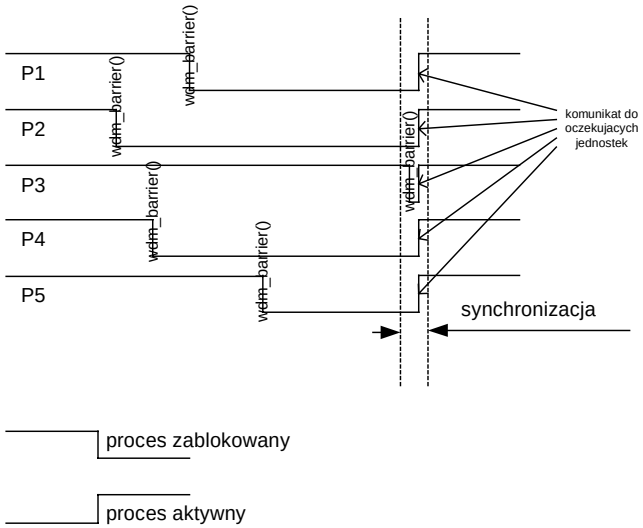
nUID - określa liczbę jednostek potrzebnych do pokonania bariery; 0 oznacza wszystkie jednostki musza wywołać wdm_barrier,

name- określa nazwę bariery; w danej chwili różne jednostki mogą być wstrzymane (zawieszone) na różnych barierach,

… - następnie należy wymienić nUID identyfikatorów jednostek.

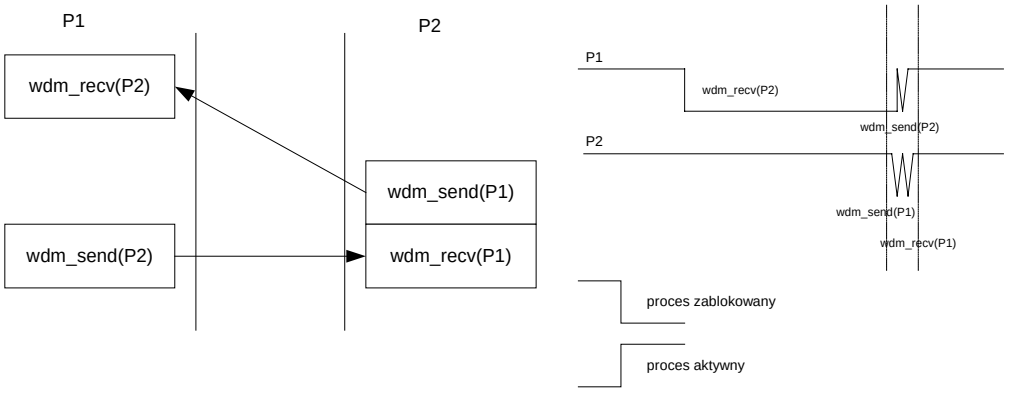
Funkcja ***wdm_barrier*** jest blokująca. W chwili wywołania ***wdm_barrier*** jednostka wysyła informację do WDMQS. Jeżeli nazwa *name* nie jest jeszcze zarejestrowana jako bariera, WDMQS rejestruje barierę i od tej chwili śledzi, które jednostki wywołały funkcję ***wdm_barrier*** z takim parametrem. Kiedy wszystkie jednostki biorące udział w barierze *name* zgłoszą swoją obecność, WDMQS wysyła do nich specjalny komunikat i usuwa barierę (Rys. 1-4). Wszystkie jednostki opuszczają funkcję ***wdm_barrier()*** i przechodzą do

wykonania kolejnych linii kodu. W czasie, gdy jednostka jest zablokowana funkcją ***wdm_barrier,*** nie wykonuje ona żadnych obliczeń i czeka na sygnał od WDMQS. Można wtedy z poziomu konsoli wydać polecenie RUN. Sprawi ono, że WDMQS usunie jednostkę z listy barier i wyśle do niej taki sam sygnał jak podczas opuszczania bariery.



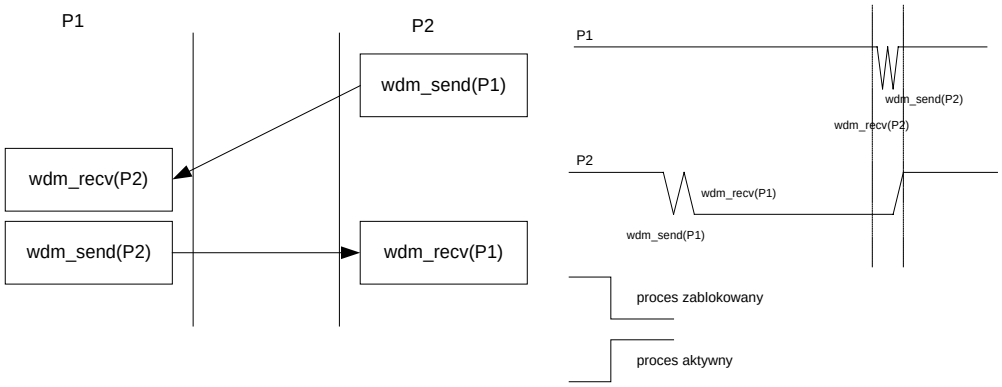
Rys. 1-4 Działanie bariery wdm_barrier

Synchronizację dwóch procesów można także uzyskać poprzez wywołanie blokującej funkcji ***wdm_recv()*** i ***wdm_send():*** przez pierwszą z jednostek ***wdm_send()*** oraz ***wdm_recv()*** przez drugą. Sytuację taką przedstawia Rys. 1-5. Proces P1 jako pierwszy dotarł do punktu synchronizacji i wywołał ***wdm_recv(P2)***. Pozostanie w tym miejscu do chwili, gdy proces P2 osiągnie punkt synchronizacji wysyłając pakiet synchronizujący. Kolejne instrukcje ***wdm_send(P2)*** i ***wdm_recv(P1)*** wykonają się niemal równocześnie.



Rys. 1-5 Synchronizacja dwóch jednostek (A)

Sytuację, w której P2 jako pierwszy osiągnie punkt synchronizacji przedstawia Rys. 1-6.



Rys. 1-6 Synchronizacja dwóch jednostek (B)

Wywołanie funkcji ***wdm_send()*** nie jest blokujące. Program nie czeka na to, aż odbiorca odbierze dane. Synchronizacja zachodzi podczas wywołania funkcji ***wdm_recv().*** Nie ma też niebezpieczeństwa wystąpienia blokady, kiedy oba procesy wywołują jednocześnie ***wdm_send(),*** a następnie ***wdm_recv()***. Blokada wystąpi, kiedy oba procesy wywołają ***wdm_recv(),*** a w ich buforach nie będzie jeszcze danych. Użycie funkcji ***wdm_barrier()*** jest jednak najbardziej zalecaną i przejrzystą metodą synchronizacji procesów.

1.3 Bezpieczeństwo

Ponieważ system pozwala zdalnie uruchamiać programy na innych komputerach, niezbędne było rozważenie aspektów bezpieczeństwa takiego działania. W przypadku Windows 95, Windows 98, które nie oferują dodatkowych zabezpieczeń, jedynym zabezpieczeniem jest hasło oraz numer portu na jakim pracuje rexec95. Możemy uruchomić kilka instancji rexec95, każdą z innym numerem portu i hasłem. Jednak nie zmienia to faktu, że uruchomiona aplikacja ma dostęp do wszystkich zasobów komputera. Jedyne co nam pozostaje to udostępniać nasze zasoby tylko zaufanym osobom.

Dużo lepsza jest sytuacja w przypadku systemów Windows NT i Windows 2000. Serwis rexecsrv możemy zainstalować na wybranym koncie. Konto może zostać tak skonfigurowane, że będzie miało dostęp tylko do wybranych zasobów komputera, np. katalog c:\tmp. W przypadku udostępniania komputerów z systemami Windows NT i Windows 2000 przy pomocy rexec95 należy pamiętać, że uruchamiane aplikacje będą miały taki sam dostęp do zasobów jak zalogowany użytkownik.

Jak widać, jedynie Windows NT i Windows 2000 dają nam możliwość bezpiecznego udostępniania swoich zasobów.

1.4 Implementacja

Wszystkie programy systemu WDM zostały napisane w języku C++. Do kompilacji został wykorzystany ***Visual C++ 6.0***. Środowisko ***Microsoft Visual Studio Enterprise Edition*** zostało wybrane głównie z uwagi na jego integrację z systemem operacyjnym. Możliwość śledzenia programów miała tutaj decydujące znaczenie. Jednak produkt firmy Microsoft nie jest łatwy w użyciu. Jedynym źródłem dokumentacji jest w tym przypadku

MSDN Library[3]. Choć dokumentacja jest obszerna nie zawsze jest wystarczająca. Nieocenione okazały się też inne bazy danych: ***TechNet***[4] oraz ***KnowledgeBase***[5]. Właśnie w KnowledgeBase znalazł się artykuł *(Article ID: Q168349 BUG: WSAEventSelect( ) Returns WSAEINVAL if Network Events is 0)* dokumentujący błąd w funkcji WSAEventSelect(). Funkcja ta została wykorzystana przy pisaniu serwisu RexecSrv. Wszystkie te pomoce są kosztowne, ale niezbędne. Na domiar złego, wcale nie wyjaśniają wszystkich problemów związanych z programowaniem w Windows. Stąd też liczne pozycje książkowe, które możemy znaleźć w bibliografii.

Programy składające się na pakiet ***Windows Distributed Machine*** zawierają łącznie około 7000 linii kodu. Z uwagi na ich rozmiary nie są prezentowane ich wydruki. Zarówno pliki źródłowe, jak i programy wynikowe, zostały dołączone do pracy na CD-ROM-ie. Tam też znajdują się przedstawiane przykładowe programy i zadania. Wszystkie programy są 32-bitowe.

[3] MSDN Library - Microsoft Developer Network Library. Dokumentacja dostępna na CD lub DVD, niezbędna w pracy z pakietem Microsoft. Zawiera opis funkcji dostępnych w Visual Basic, Visual C++, Visual FoxPro oraz Visual InterDev. MSDN zawiera opis architektury systemu Windows, wskazówki dla programistów, opis udokumentowanych i potwierdzonych błędów w systemie.

[4] Microsoft TechNet – techniczne informacje dotyczące produktów Microsoft

[5] Microsoft Knowledge Base - baza danych z licznymi artykułami poświęconymi Windows NT, Windows 95, Microsoft Office, SMS, SQL Server, VB, IIS, IE, Encarta, Age of Empires

2 Instalacja i tworzenie rozproszonych aplikacji przy pomocy systemu i biblioteki WDM

W rozdziale tym przedstawione będą kolejne kroki jakie należy wykonać, aby korzystać z systemu i biblioteki Windows Distributed Machine. Przy omawianiu poszczególnych funkcji bibliotecznych będą przytaczane przykłady, dzięki którym łatwiej będzie można prześledzić kompletny proces tworzenia aplikacji rozproszonej pracującej z systemem WDM.

2.1 Uruchomienie komponentów

Zanim napiszemy i uruchomimy pierwszy program wykorzystujący bibliotekę WDM musimy zainstalować niezbędne komponenty systemu:

- WDMQS - instalujemy na jednym z komputerów
- Serwisy pozwalające zdalnie uruchamiać aplikacje (rexecsrv lub rexec95) - instalujemy na wszystkich komputerach biorących udział w obliczeniach lub symulacji

Uwaga: *Serwisy nie są niezbędne do działania systemu, gdyż nasze jednostki możemy uruchamiać ręcznie z linii poleceń. Jednak ich zainstalowanie uprości proces uruchamiania. Kiedy nasz program będzie gotowy i będziemy chcieli uruchomić jednostki na różnych maszynach, instalację serwisów trzeba przeprowadzić na każdym komputerze z osobna.* ***rexecsrv*** *ma tą przewagę nad* ***rexec95****, że pracuje jako serwis NT i może wystartować nawet wtedy, gdy na komputerze nikt nie jest zalogowany. W połączeniu z techniką Wake On Lan[6] możemy sobie wyobrazić, że w dowolnym momencie uruchamiamy odległe komputery z zainstalowanym serwisem. Przeprowadzamy nasze obliczenie i następnie możemy je zdalnie wyłączyć.*

2.1.1 Instalacja RexecSrv i Rexec95

RexecSrv (Windows NT, Windows 2000)

RexecSrc składa się z dwóch elementów rexecsrv.exe i rexecsrv.dll. Instalację przeprowadzamy po zalogowaniu się na konto użytkownika, który ma prawa lokalnego administratora. Oba pliki należy skopiować do dowolnego katalogu, jednak najlepiej do %SystemRoot%\system32. Następnie z menu Start->Run uruchamiamy cmd. W kolejnym

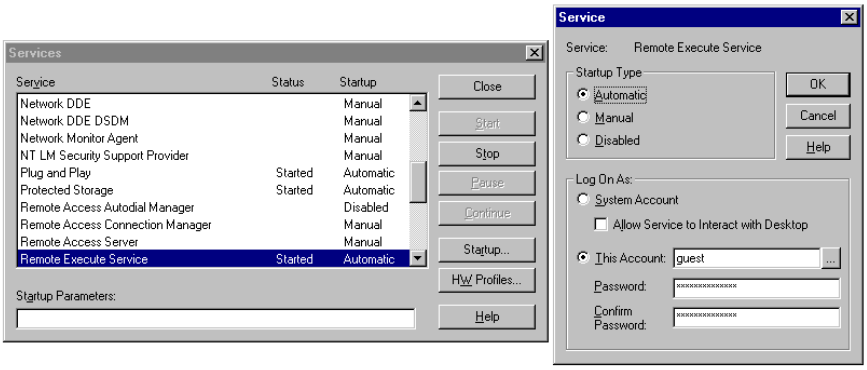
[6] *Wake on Lan – zasilanie w komputerach PC (COMPAQ, DELL, HP) jest dziś włączane elektronicznie, a karty sieciowe potrafią odebrać sygnał skierowany do komputera i włączyć zasilanie. W ten sposób możemy zdalnie włączyć komputer. Mechanizm ten jest jeszcze ograniczony tylko do sieci lokalnej. Komputer wybieramy podając MAC (Media Access Control) adres karty sieciowej.*

kroku z linii poleceń uruchamiamy ***rexecsrv.exe –install.*** W trakcie instalacji program dokonuje niezbędnego wpisu w rejestry. Wpis ten jest konieczny, aby program *Event Viewer* poprawnie interpretował komunikaty od serwisu.

Remote Execute Service dokonuje wpisu w rejestrach w kluczu:

`(HKLM\SYSTEM\CurrentControlSet\Services\EventLog\Application\RemoteExecuteService)`

Po zainstalowaniu programu należy przydzielić mu konto, na którym ma pracować i wybrać sposób uruchamiania (manual lub automatic Rys. 2-1).

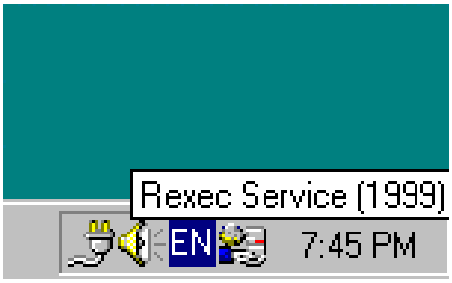


Rys. 2-1 Konfiguracja Remote Execute Service

Serwis rexecsrv możemy w każdej chwili usunąć pisząc polecenie: ***rexecsrv –remove***. Program sam usunie wpisy z rejestrów. Jeżeli w trakcie pracy napotkamy na problemy, pomocny będzie program *Event Viewer,* za pomocą którego można obejrzeć komunikaty zapisywane przez serwis.

Rexec95 (Windows 95, Windows 98, Windows NT Windows 2000)

Rexec95 uruchamiamy tak jak każdą aplikację w Windows. Podając odpowiedni parametr możemy zdecydować jaki port chcemy wykorzystać. Program rexec95 możemy uruchomić kilka razy podając zawsze inny numer portu. Jeżeli udostępniamy nasz komputer różnym osobom, każda z nich może korzystać z innego portu.



Rys. 2-2 Ikona Rexec95 widoczna na pasku zadań

Po uruchomieniu programu w prawym dolnym rogu ekranu na pasku zadań pojawia się ikona przedstawiona na Rys. 2-2. Podwójne przyciśnięcie klawisza myszy na ikonie pozwala podglądać log aplikacji, w którym możemy zobaczyć kto i co uruchamia na naszym komputerze.

Wskazówki:

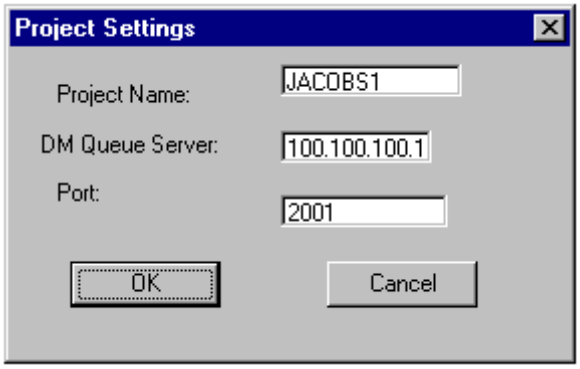
- *Jeżeli korzystamy z systemu Windows NT lub Windows 2000 warto w pierwszej fazie pisania aplikacji skorzystać z programu rexec95. Uruchamiane jednostki będą pojawiać się na ekranie i będziemy mogli od razu zauważyć ewentualne błędy. Dopiero kiedy zobaczymy, że aplikacja zachowuje się poprawnie, możemy zacząć korzystać z serwisu rexecsrv.*
- *Bardzo wygodne jest uruchomienie na jednym komputerze obu serwisów. Rexecsrv na porcie 1999 i rexec95 na porcie 2000. Wtedy wpisując odpowiedni port we właściwościach jednostki (WDMConsole) możemy kontrolować sposób jej uruchamiania. W zależności od potrzeb jedne jednostki mogą się pojawiać na ekranie, a inne nie.*

2.1.2 Uruchomienie WDMQS

Jak już wiemy z rozdziału 1.1, gdzie została omówiona budowa systemu, cała komunikacja odbywa się za pośrednictwem jądra *WDMQS*. Komponent ten musi zostać uruchomiony przed przystąpieniem do obliczeń lub symulacji. Nie ma znaczenia, na którym z komputerów będzie działał WDMQS. Program uruchamiamy z linii poleceń lub instalujemy jako serwis. Uruchamiając program możemy decydować, na którym porcie ma on pracować. Domyślnie jest to port 2001. Znajomość adresu ip i portu WDMQS będzie nam potrzebna podczas uruchamiania jednostek.

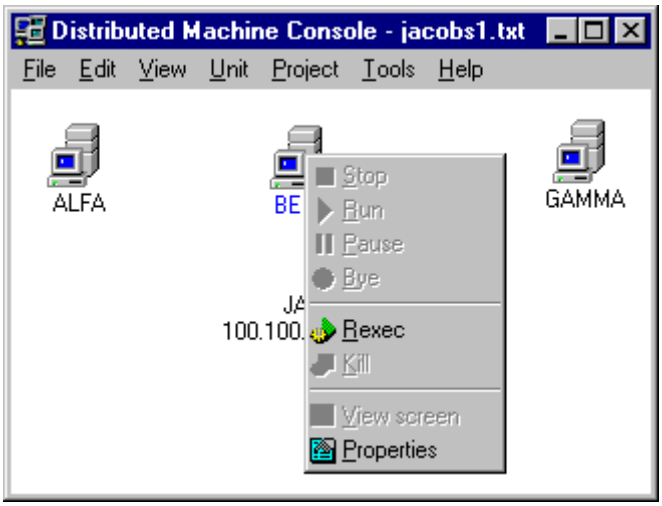
2.2 Definiowanie struktury za pomocą WDM Console

Pierwszym krokiem po uruchomieniu WDMConsole jest zdefiniowanie nazwy projektu oraz adres WDMQS. Nazwa musi być unikalna. W WDMQS może być zarejestrowanych kilka projektów. Tak więc wchodzimy do menu Project->Settings Rys. 2-3.



Rys. 2-3 Parametry projektu

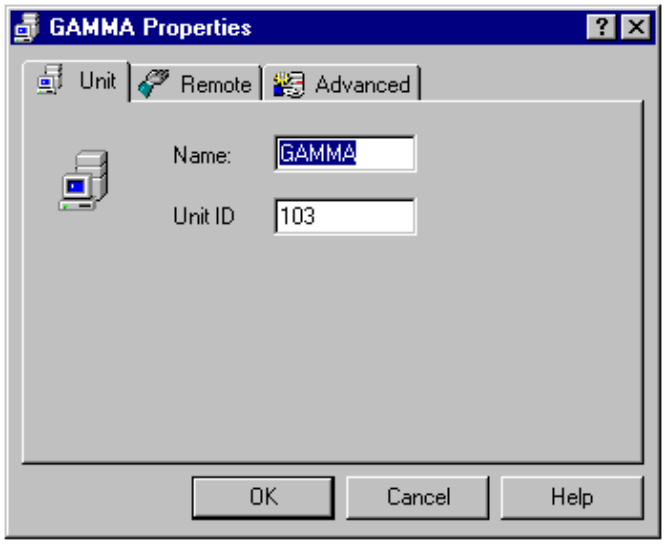
Kolejny krok to definiowanie jednostek biorących udział w obliczeniach. Nowe jednostki dodajemy do projektu klawiszem <INSERT>. Kiedy stworzymy odpowiednią liczbę jednostek musimy określić ich parametry (*Properties* Rys. 2-4*)*.



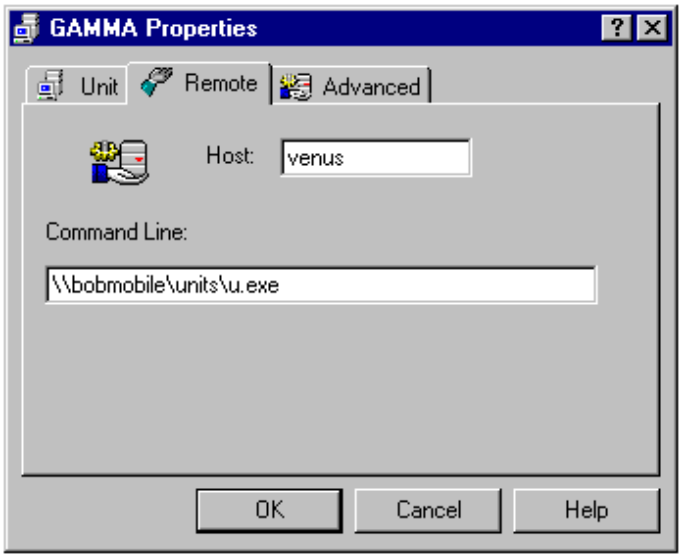
Rys. 2-4 Jednostki zdefiniowane w WDMConsole

- ***Name*** posłuży nam jako parametr podczas uruchamiania jednostek. Po uruchomieniu jednostka połączy się z WDMQS podając name i project. W efekcie QDMQS zwróci jej UNIT ID.
- ***Unit ID*** – liczba całkowita z zakresu <1-254>. UID wykorzystamy w programie jednostki do identyfikacji właściwej sekcji kodu, która ma zostać wykonana.
- ***Host*** – to pole wykorzysta WDMConsole do komunikacji z serwisem wywołującym zdalnie program jednostki.
- ***Command line*** – ten parametr zostanie przekazany do rexecsrv lub rexec95, a następnie serwis podejmie próbę uruchomienia właściwego programu.
- ***PORT*** – port, pod którym pracuje rexec95. Serwis może pracować na różnych portach w zależności od komputera.
- ***User name i Password*** – dodatkowo dostęp to rexec95 lub sexecsrv może być zabezpieczony hasłem.

W celu podania parametrów jednostki przyciskamy prawy klawisz myszy na ikonie jednostki i wybieramy *Properties* Rys. 2-4. To samo możemy osiągnąć zaznaczając jednostkę i wybierając *Properties* z menu *Unit.* Kolejne kroki definiowania parametrów jednostki przedstawione są na rysunkach poniżej (Rys. 2-5, Rys. 2-6).



Rys. 2-5 Parametry podstawowe jednostki GAMMA



Rys. 2-6 Parametry zdalnego uruchomienia jednostki GAMMA

Jeżeli mamy już zdefiniowany projekt należy zarejestrować go w WDMQS i możemy teraz przystąpić do obliczeń. Jednostki uruchamiamy ręcznie na dowolnym komputerze podając jako parametry dane WDMQS i projektu np.:

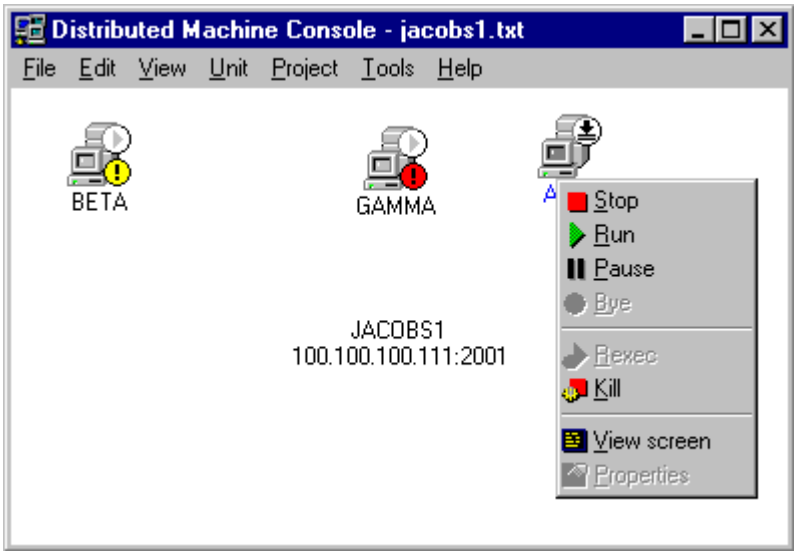
unit.exe bobmobile 2001 JACOBS1 BETA

2.3 Zdalne uruchomienie jednostki za pomocą WDM Console

WDMConsole potrafi komunikować się z serwisami *rexec95* i *rexecsrv*. Za jego pomocą można uruchomić dowolny program na odległym komputerze pod warunkiem, że jest tam uprzednio zainstalowany serwis *rexec* i znamy jego parametry *(port, username, password).* Sprawdzimy teraz, jak działa zdalne uruchamianie. Spróbujemy uruchomić na odległej maszynie program kalkulator graficzny calc.exe. Jednak, aby program pokazał się

na ekranie, musi tam pracować *rexecsrv* na koncie *System Account* lub *rexec95*. Jeżeli rexecsrv pracuje na innym koncie, calc.exe pojawi się jako proces pracujący w tle bez dostępu do konsoli. Tak wiec z menu Tools wybieramy Rexec i podajemy nazwę lub adres ip komputera, na którym chcemy uruchomić program. W polu command line wpisujemy calc.exe i wciskamy klawisz OK. Na odległym komputerze powinien pojawić się kalkulator. Tak uruchomiony proces możemy również zdalnie zabić: Tools-> Kill. Kalkulator powinien zniknąć. W ten sposób sprawdziliśmy działanie zdalnego uruchamiania.

Możemy przystąpić do uruchamiania naszych jednostek. W tym miejscu zakładamy, że mamy już napisany program jednostki. Szczegółowe instrukcje jak napisać taki program znajdują się w dalszej części tego rozdziału. Aby uruchomić jednostkę zaznaczamy jej ikonę i wciskamy prawy klawisz myszy. Z menu wybieramy opcję *run*. Rys. 2-7 przedstawia kilka efektów takiego wywołania. W przypadku kłopotów z uruchomieniem pomocny może być dodatkowy log (menu View->TraceWindow).



Rys. 2-7 Zdalne uruchamianie jednostek

Oto jak należy interpretować stan jednostek:

- ***ALFA*** – jednostka została uruchomiona, skomunikowała się z WDMQS i wywołała funkcję wdm_barrier.
- ***BETA*** – próba remote execute powiodła się (proces został uruchomiony). Jednak jednostka nie skomunikowała się w czasie 5s z WDMQS. Najczęściej powodem takiego zachowania jest wpisanie w projekcie nazwy komputera, na którym pracuje WDMQS zamiast jego adresu IP. Jeżeli nazwa nie jest właściwie rozpoznawana przez komputer, na którym uruchomiona została jednostka, nie może się ona skomunikować z WDMQS *(np. localhost, albo nazwa, która jest tylko w naszym pliku hosts)*. Inna przyczyna to błąd w programie jednostki, która z jakiegoś powodu nie wywołała funkcji wdm_Unit_Register();
- ***GAMMA*** – próba *rexemote execute* nie powiodła się. *View -> Trace Window* zawiera dodatkowe informacje. Przyczyna: na odległym komputerze nie ma serwisu *rexec*, niewłaściwy jest port albo hasło. Jednak najczęściej rexecsrv nie może uruchomić programu, który podaliśmy we właściwościach jednostki w polu w *command line*.

2.4 Biblioteka WDM i jej wykorzystanie w przykładowych programach jednostki

Program jednostki piszemy w C/C++ i kompilujemy kompilatorem ***Microsoft Visual C++ 6.0***. Funkcje biblioteczne zadeklarowane są w pliku ulib.h, a podczas konsolidacji należy dołączyć ulib.lib oraz wsock32.lib. Omawiając poszczególne funkcje biblioteki zaprezentujemy ich przykładowe wykorzystanie. Ogólna struktura programu jednostki wygląda najczęściej następująco:

```
{
   //Rejestracja w WDMQS
   UID = wdm_UnitRegister(…)

   //odebranie danych od jednostki typu master
   wdm_recv(…);
   //wykonanie kodu jednostki
   //wlasciwa sekcja jest identyfikowana na podstawie UID
   . . .

   //przeslanie wynikow do jednostki typu master
   wdm_send(…);
   //powiadomienie WDMQS o zakonczeniu obliczen
   wdm_UnitStop()
}
```

Program może być wspólny dla wszystkich jednostek (podejście SPMD). Podczas rejestracji w WDMQS jednostka dowiaduje się jaki jest jej identyfikator *UID* (jaką część kodu ma wykonać lub z jakimi danymi). Jednak najczęściej piszemy program jednostek i program jednostki typu MASTER. Zadaniem jednostki ***master*** jest rozdysponowanie danych i odebranie wyników od jednostek typu ***slave***. Jest to więc model klient-serwer, gdzie serwerem jest jednostka obliczeniowa (***slave)*** a klientem jednostka ***master***.

2.4.1 Funkcje rejestrujące

*WORD wdm_UnitRegister (int argc, char **argv)*

Zadaniem funkcji jest zarejestrowanie jednostki obliczeniowej w ***WDMQS (Windows Distributed Machine Queue Server)*** – module komunikacyjnym. Informacje o WDMQS jednostka otrzymuje jako argumenty wywołania programu.

W przypadku, gdy jednostka została uruchomiona ręcznie, serwer nic o niej nie wie. Jeżeli jest ona uruchamiana za pomocą serwisu *rexecsrv,* to i tak nie można przewidzieć jaki port otrzyma. Jednostka nie może mieć z góry przyznanego portu, bo wtedy nie uruchomilibyśmy kilku na jednym komputerze. Z tych powodów, to jednostka musi zainicjować komunikację z ***WDMQS***. Od momentu rejestracji serwer będzie wiedział gdzie jest i jak się z nią komunikować. Parametry serwera i projektu to jedyny sposób, aby powiadomić jednostkę, gdzie znajduje się W*DMQS i jaką* funkcje ma pełnić. Jeżeli jednostka jest uruchamiana za pomocą konsoli (***WDMConsole***) właściwe argumenty (ip, port, project, name) są przekazane automatycznie. W przypadku ręcznego uruchamiania, parametry musimy podać sami. Funkcja **wdm_*UnitRegister()*** zwraca identyfikator jednostki w WDMQS. Po wykonaniu tej funkcji na ekranie konsoli powinna pojawić się ikona:

 - jednostka aktywna *(wywołano wdm_UnitRegister)*

Taka ikona może pojawić się tylko na chwilę, gdyż jednostka zaraz przejdzie do wykonania kolejnych instrukcji typu: ***wdm_recv()*** lub ***wdm_barrier()***.

void wdm_UnitStop ()

Po zakończeniu obliczeń jednostka powinna jako ostatnią funkcję wywołać ***wdm_UnitStop().*** W ten sposób jednostka powiadamia *WDMQS* o zakończeniu pracy i pozostaje w tym stanie oczekując na sygnał “bye” z konsoli. Stan ten symbolizuje ikona:

 - jednostka zakończyła obliczenia *(wdm_UnitStop)*

A oto przykład programu:

```
//sample1.cpp
#include "ulib.h"
#define ALFA  101
#define BETA  102
#define GAMMA 103
void main(int argc, char **argv)
{
int UID;
int x,y,z;
   UID = wdm_UnitRegister(argc, argv);
   switch (UID)
   {
   case ALFA: // kod jednostki ALFA
      x = 10;
      break;
   case BETA: // kod jednostki BETA
      y = 100;
      break;
   case GAMMA: // kod jednostki GAMMA
      z = 110;
      break;
   }
   wdm_UnitStop();
}
```

2.4.2 Funkcje komunikacyjne

Przykład z poprzedniego rozdziału (*sample1.cpp*) pokazuje jak jednostka rozpoznaje, którą część kodu ma wykonać. To już pozwala napisać taki program, który będziemy mogli wykonać na kilku komputerach. Jednak kluczową kwestią w tej pracy jest komunikacja między rozproszonymi jednostkami.

Przesyłanie danych i synchronizację zapewnia zestaw funkcji podany poniżej:

```
int  wdm_recv(int src , char  *ptr);
int  wdm_recv_any(WORD *rsrc, char *ptr);
int  wdm_async_recv(int src , char  *ptr);
```

```
int  wdm_async_recv_any(WORD *rsrc, char *ptr);
int  wdm_send(int dst , char  *ptr, int len);
int  wdm_send_over(int dst , char  *ptr, int len);
int  wdm_broadcast(char  *ptr, int len);
int  wdm_broadcast_over(char  *ptr, int len);
void wdm_barrier(int nUID, char *name, ...);
void wdm_barrier_all(char *name);
```

W pierwszej części zostaną przedstawione trzy podstawowe funkcje **wdm*_barrier(),* wdm*_recv()* i **wdm***_send()*, które pozwalają na synchronizację i wymianę danych między jednostkami oraz przykład programu z wykorzystaniem tych funkcji. Dopiero później zostaną omówione pozostałe funkcje z tej sekcji.

*int wdm_recv(int src , char *ptr)*

src identyfikator nadawcy

ptr wskazanie na bufor z danymi

Funkcja zwraca ilość otrzymanych bajtów. Jednostka wywołuje ***wdm_recv(ALFA, &x)*** kiedy chce odebrać dane z jednostki ALFA. Jest to funkcja blokująca. Taki stan jednostki jest reprezentowany w WDMConsole przez odpowiednią ikonę. Następna instrukcja zostanie wykonana dopiero po odebraniu żądanego pakietu. Funkcja ***wdm_recv()*** pozwala w prosty sposób na synchronizację jednostek *(ten mechanizm został omówiony w rozdziale **1.2.1 Synchronizacja procesów**)*.

 - jednostka oczekuje blokująco na dane od innej jednostki *(wdm_recv)*

*int wdm_send(int dst , char *ptr, int len)*

dst identyfikator odbiorcy

ptr wskazanie na bufor z danymi

len rozmiar danych

Funkcja ***wdm_send*** służy do wysyłania danych. Dane są wysyłane natychmiast i trafiają do bufora wejściowego jednostki *dst*. Będą tam czekały do momentu, aż jednostka *dst* zgłosi na nie zapotrzebowanie wywołując ***wdm_recv***. ***wdm_send()*** zwraca ilość wysłanych bajtów.

*void wdm_barrier(int nUID, char *name, ...)*

nUID liczba jednostek niezbędnych do pokonania bariery

name nazwa bariery

… identyfikatory jednostek

Funkcja ***wdm_barrier*** służy do synchronizacji procesów jednostek. Znajduje ona swoje zastosowanie zwłaszcza podczas obliczeń synchronicznych. Parametr ***nUID*** określa liczbę zablokowanych jednostek. Parametr ***name*** określa nazwę bariery. Następnie należy wymienić ***nUID*** identyfikatorów jednostek. Rozważając program *sample1.cpp* zauważamy, że kod jednostek nie zawiera żadnych mechanizmów synchronizacji. Może się zatem zdarzyć, że kod jednostki ALFA wykona się do końca, jeszcze zanim zostaną uruchomione

kolejne jednostki. Wykorzystując funkcję ***wdm_barrier*** w naszym przykładzie kod funkcji *main()* wyglądałby tak jak w programie *sample2.cpp*.

```
// sample2.cpp
void main(int argc, char **argv)
{
int UID;
int x,y,z;

   UID = wdm_UnitRegister(argc, argv);
   switch (UID)
   {
    . . .
    . . .
   }
   wdm_barrier(3, "STOP", ALFA, BETA, GAMMA);
   wdm_UnitStop();
}
```

Po uruchomieniu wszystkie jednostki skomunikują się z WDMQS, a następnie, wywołują funkcję ***wdm_barrier***. Każda jednostka będzie trwała w tym stanie do momentu, kiedy WDMQS stwierdzi, że wszystkie jednostki wymienione w wywołaniu funkcji osiągnęły barierę, która identyfikowana jest przez nazwę *("START")*. Wtedy WDMQS powiadomi jednostki, a nasz program (programy) przejdzie do kolejnych instrukcji. Przebywanie jednostki w stanie „*barrier*" jest obrazowane na konsoli w postaci ikony:

 - jednostka wywołała funkcję *wdm_barier()*

Sample3.cpp jest przykładem aplikacji rozproszonej wykorzystującej poznane do tej pory funkcje i mechanizmy. Aplikacja składa się z 3 jednostek *ALFA, BETA, GAMMA*. Jednostki *ALFA* i *BETA* mają za zadanie obliczyć wartości x oraz y i następnie przesłać je do jednostki *GAMMA.* Jednostka *GAMMA* wykona z kolei operacje dodawania i wyświetli wyniki na ekranie. Tak może wyglądać nasza pierwsza aplikacja rozproszona pracująca z *systemem Windows Distributed Machine*.

```
//sample3.cpp
//prosty przyklad uzycia funkcji wdm_send i wdm_recv
#include "ulib.h"
#define ALFA  101
#define BETA  102
#define GAMMA 103
void main(int argc, char **argv)
{
int UID;
int x,y,z;

   UID = wdm_UnitRegister(argc, argv);

   switch (UID)
```

```
   {
   case ALFA: // kod jednostki ALFA
      x = 10;
      wdm_barrier(3, "DANE", ALFA, BETA, GAMMA);
      wdm_send(GAMMA, &x, sizeof(x));
      break;
   case BETA: /* kod jednostki BETA
      y = 100;
      wdm_barrier(3, "DANE", ALFA, BETA, GAMMA);
      wdm_send(GAMMA, &y, sizeof(y));
      break;
   case GAMMA: // kod jednostki GAMMA
      wdm_barrier(3, "DANE", ALFA, BETA, GAMMA);
      wdm_recv(ALFA, &x);
      wdm_recv(BETA, &y);
      z = x + y;
      printf("z = %d\n",z);
      break;
   }
   wdm_UnitStop();
}
```

*int wdm_recv_any(WORD *rsrc, char *ptr);*

rsrc w tej zmiennej zostanie umieszczony identyfikator z nadawcy

ptr wskazanie na bufor, w którym zostaną umieszczone dane

Funkcja ta pozwala odebrać dane od dowolnej jednostki. Pakiety są kolejkowane przez WDMQS. Funkcja zwraca liczbę otrzymanych bajtów. Po odebraniu danych ***rsrc*** zawiera identyfikator jednostki, która nadesłała dane.

*int wdm_async_recv(int src , char *ptr);*

src identyfikator nadawcy

ptr wskazanie na bufor z danymi

Funkcja sprawdza, czy w jej buforze wejściowym są jakiś dane od jednostki ***src***. Jeżeli tak, to odbiera pierwszy pakiet i zwraca ilość otrzymanych bajtów, w przeciwnym razie, zwraca wartość 0.

*int wdm_async_recv_any(WORD *rsrc, char *ptr);*

rsrc w tej zmiennej zostanie umieszczony identyfikator z nadawcy

ptr wskazanie na bufor w którym zostaną umieszczone dane

Funkcja ***wdm_async_recv_any*** analogicznie do ***wdm_async_recv***, jednak nie ma znaczenia, kto jest nadawcą pakietu. Po wywołaniu, zwraca ilość otrzymanych bajtów lub 0.

Obie funkcje ***wdm_async_recv i wdm_async_recv_any*** działają podobnie jak już omówione ***wdm_recv i wdm_recv_any*** z tą różnicą, że nie powodują blokowania jednostki.

Jeżeli nie ma żadnych danych w jej buforze wejściowym, funkcja zwraca 0 i program przechodzi do wykonania kolejnych instrukcji.

int wdm_broadcast(char *ptr, int len);

len rozmiar danych

ptr wskazanie na bufor, w którym znajdują się dane

Funkcja zwraca ilość wysłanych bajtów. Użycie ***wdm_broadcast()*** pozwala jednocześnie wysłać dane do wszystkich jednostek. Dane nie są wysyłane do jednostki, która wywołała funkcję.

int wdm_send_over(int dst , char *ptr, int len);

int wdm_broadcast_over(char *ptr, int len);

Funkcje ***wdm_send_over*** i ***wdm_broadcast_over*** działają analogicznie do poznanych już ***wdm_send*** i ***wdm_broadcast***. Różnica polega na tym, że pakiet wysłany za pomocą takiej funkcji trafiając do bufora wejściowego odbiorcy zastępuje ostatnio wysłany pakiet. Funkcja ta znajduje swoje zastosowanie w przypadku obliczeń asynchronicznych. Wolniejsza jednostka nie jest w stanie konsumować wszystkich nadsyłanych danych, a interesują ją tylko najnowsze dane. Dzięki tej funkcji szybsze jednostki nie muszą czekać na wolniejsze i jednocześnie wysyłane przez nie dane nie powodują przepełnienia buforów wejściowych jednostek wolniejszych. Wolniejsze jednostki mają zawsze aktualne dane.

void wdm_barrier_all(char *name);

Jest to odmiana ***wdm_barier()*** Stosujemy ją wtedy, gdy barierę muszą pokonać wszystkie jednostki i nie chcemy ich wymieniać

2.4.3 Funkcje wejścia, wyjścia

Są to bardzo użyteczne i niespotykane w innych systemach tego typu funkcje. Realizują te same funkcje co ich odpowiedniki w języku *C*, a w tle komunikują się z jądrem WDMQS.

```
void wdm_printf(char *fmt, ...);
void wdm_scanf(char *s);
```

Funkcje wejścia, wyjścia pozwalają na interaktywną pracę z odległymi jednostkami. Dzięki nim możemy obserwować wyniki pośrednie obliczeń będąc fizycznie przy konsoli oraz wprowadzać dane, o które proszą jednostki. Funkcji tych nie należy nadużywać, gdyż mogą generować duży ruch w sieci.

void wdm_scanf(char *s);

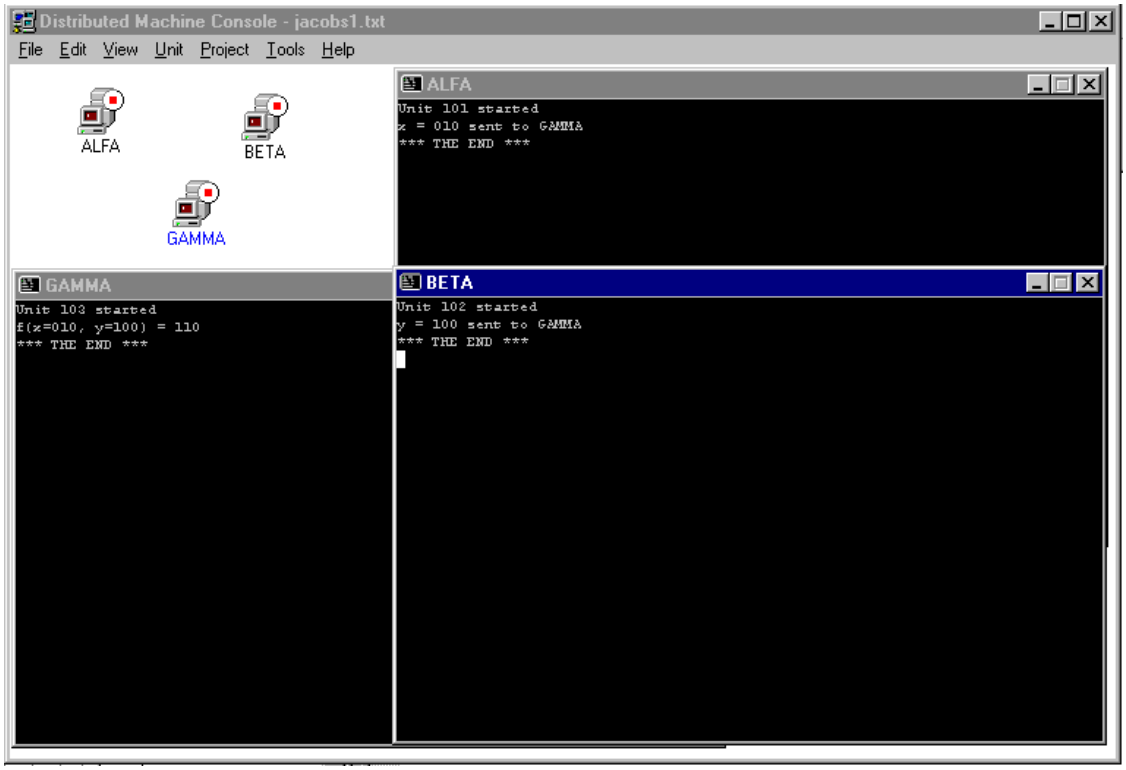
Funkcja ***wdm_scanf()*** pozwala operatorowi konsoli na podanie z klawiatury danych, o które prosi jednostka na odległym komputerze. Dane można wprowadzać w obu miejscach tzn. na konsoli i na komputerze, na którym pracuje jednostka.

 - jednostka oczekuje na wprowadzenie danych *(wdm_scanf)*

void wdm_printf(char *fmt, ...);

Jest to odpowiednik funkcji *printf()* w języku *C*. ***wdm_printf()*** drukuje wyniki na swoim ekranie i przesyła je do konsoli.

Program *sample4.cpp* przedstawia jak za pomocą funkcji ***wdm_printf()*** możliwa jest prezentacja wyników pośrednich, na ekranie WDMConsole.



Rys. 2-8 Wyniki działania programu sample4.cpp

```
//sample4.cpp
// prosty przyklad uzycia funkcji wdm_send i wdm_recv
// oraz wdm_printf
#include "..\ulib\ulib.h"
#define ALFA  101
#define BETA  102
#define GAMMA 103
void main(int argc, char **argv)
{
```

```
int UID;
int x,y,z;

   UID = wdm_UnitRegister(argc, argv);
   wdm_printf("Unit %03d started\n", UID);
   wdm_barrier(3, "START", ALFA, BETA, GAMMA);
   switch (UID)
   {
   case ALFA:       // kod jednostki ALFA
      x = 10;
      wdm_send(GAMMA,(char *) &x, sizeof(x));
      wdm_printf("x = %03d sent to GAMMA\n", x);
      break;
   case BETA:      // kod jednostki BETA
      y = 100;
      wdm_send(GAMMA,(char *) &y, sizeof(y));
      wdm_printf("y = %03d sent to GAMMA\n", y);
      break;
   case GAMMA:    // kod jednostki BETA
      wdm_recv(ALFA,(char *) &x);
      wdm_recv(BETA,(char *) &y);
      z = x + y;
      wdm_printf("f(x=%03d, y=%03d) = %03d \n", x, y, z);
      break;
   }
   wdm_printf("*** THE END ***\n");
   wdm_UnitStop();
   return;
}
```

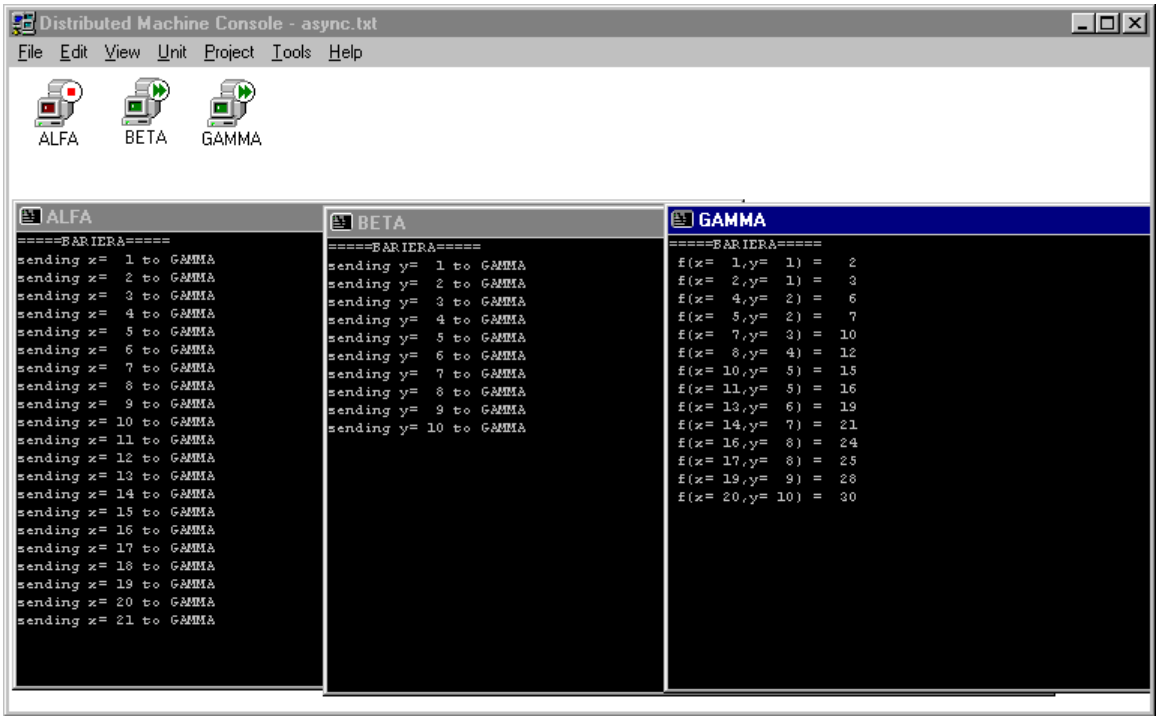
Po uruchomieniu wszystkie jednostki synchronizują się za pomocą bariery ***wdm_barrier()***. Następnie każda z nich wykonuje swoją cześć obliczeń. Identyfikator ***UID*** został im przydzielony podczas rejestracji w WDMQS. Jednostki ALFA i BETA wysyłają dane przy pomocy funkcji ***wdm_send()*** do jednostki GAMMA. Dane odbierane są za pomocą funkcji ***wdm_recv()***. Wszystkie jednostki wypisują na ekranie swoje wyniki za pomocą funkcji ***wdm_printf().*** Użycie tej funkcji powoduje przesłanie informacji, która pojawia się na ekranie jednostki do WDMQS, a następnie do WDMConsole. Na Rys. 2-8 widzimy, że zawartość ekranu jednostek możemy obserwować, korzystając z graficznego interfejsu WDMConsole.

Na zakończenie przykład programu realizującego obliczenia w sposób asynchroniczny *(sample5.cpp).*

```
// sample5.cpp Obliczenia asynchroniczne
#include "..\ulib\ulib.h"
#define ALFA  101
#define BETA  102
#define GAMMA 103
void main(int argc, char **argv)
{
int UID;
```

```
int x=0,y=0,z=0;
   UID = wdm_UnitRegister(argc, argv);
   wdm_printf("=====BARIERA=====\n");
   wdm_barrier(3, "START", ALFA, BETA, GAMMA);
   int i=20;
   do
      switch (UID)
      {
      case ALFA:
         x++;
         wdm_printf("sending x=%3d to GAMMA\n",x);
         wdm_send_over(GAMMA, (char *)&x, sizeof(x));
         break;
      case BETA:
         y++;
         wdm_printf("sending y=%3d to GAMMA\n",y);
         wdm_send_over(GAMMA, (char *)&y, sizeof(y));
         break;
      case GAMMA:
         wdm_async_recv(ALFA, (char *) &x);
         wdm_async_recv(BETA, (char *) &y);
         z = x + y;
         wdm_printf(" f(x=%3d,y=%3d) = %3d\n",x,y,z);
         break;
      }
   while (i-->0);
   wdm_UnitStop();
}
```

Każda z jednostek została uruchomiona na innym komputerze (Pentium 133Mhz, Pentium MMX 230Mhz, Pentium II 333Mhz). W pętli wykonywane są kolejne iteracje. Jednostki ALFA i BETA wysyłają swoje wyniki do jednostki GAMMA przy pomocy funkcji ***wdm_send_over***. Jeżeli w buforze wejściowym jednostki GAMMA znajduje się pakiet od jednostki ALFA, który nie został jeszcze odebrany nowy pakiet zastępuje jego miejsce. Tym sposobem nieaktualny już pakiet jest tracony. Jak widać na Rys. 2-9 pakiety od jednostki ALFA są generowane tak szybko, że GAMMA nie nadąża z ich konsumowaniem. Nie zdążyła odebrać pakietów 3, 6, 9, 12, 15, 18. Widać też, że jednostka ALFA jako pierwsza zakończyła obliczenia.



Rys. 2-9 Rezultat działania programu sample5.cpp

2.4.4 Funkcje dodatkowe

```
int  wdm_kill(char *host, WORD port,
              DWORD PID, char *username, char *password);
int wdm_rexec(char *host, WORD port,
              char *cmdline, char *username, char *password);
void wdm_ParseArgs(int argc, char **argv, char *ip,
                   WORD *port, char *project, char *name);
```

int wdm_rexec(char *host, WORD port, char *cmdline, char *username, char *password);

Funkcja ***wdm_rexec*** służy do komunikacji z *Remote Execute Service* i uruchamiania procesów na innych maszynach. Funkcja zwraca deskryptor utworzonego procesu PID lub zero w przypadku niepowodzenia.

int wdm_kill(char *host, WORD port, DWORD PID, char *username, char *password);

Deskryptor procesu otrzymany w wyniku wywołania *wdm_rexec* możemy później wykorzystać do zabicia procesu. Do tego celu służy funkcja *wdm_kill.*

Za pomocą funkcji ***wdm_kill*** i ***wdm_rexec*** programista może sam z wnętrza swojego programu wywoływać programy na innych komputerach. Funkcje komunikują się z pracującymi tam serwisami rexec95 i rexecsrv przekazując im zlecenia.

void wdm_ParseArgs(int argc, char **argv, char *host, WORD *port, char *project, char *name);

Funkcja zwraca host, port, project oraz name, które zostały przekazane jako argumenty wywołania programu. Programista może je wykorzystać do własnych potrzeb.