ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1995
Seria: INFORMATYKA z. 28 Nr kol. 1270

Rafał KRAJEWSKI*

PRZETWARZANIE ROZPROSZONE W SIECI NOVELL NETWARE – BIBLIOTEKA FUNKCJI**

Streszczenie. Opracowanie to jest uzupełnieniem artykułu [11] i przedstawia zbiór narzędzi programowych pozwalających na stworzenie środowiska przetwarzania rozproszonego opartego na lokalnej sieci komputerowej NetWare firmy Novell. Przedstawiona biblioteka funkcji umożliwia tworzenie aplikacji w języku wysokiego poziomu, w której istnieje możliwość generowania procesów potomnych i ich zdalnego uruchamiania na stacjach roboczych, a także wymiany informacji między procesami oraz synchronizacji procesów.

DISTRIBUTED PROCESSING ON NOVELL'S NETWARE NETWORK SYSTEM – LIBRARY FUNCTIONS

Summary. This paper is an appendix to [11] and presents a set of software tools for the creation of a distributed processing environment, based on a Novell NetWare local area network. A library of functions for a high-level language (C) is presented. Such a library an application to be created with the possibilities of generating child-processes, to execute them remotely on workstations, and to implement information exchange and synchronisation between processes.

*Instytut Informatyki Teoretycznej i Stosowanej PAN, 44-100 Gliwice, ul. Bałtycka 5, rafal@zhsk2.iitis.polsl.gliwice.pl.

**Pracę wykonano w ramach grantu KBN nr 3 P406 011 04.

TRAITEMENT DISTRIBUE DANS LES SYSTEMES RESEAUX NETWARE – BIBLIOTHEQUE DE FONCTIONS

Résumé. Le présent travail compléte l'article [11] en décrivant un ensemble d'utils logiciels permettants de créer un environnement de traitement distribué basé sur un reseau local NetWare de Novell. La librairie de fonctions presentée permet de créer, dans une langue de haut niveau, une application munie de la possibilité de générer des processus "enfants" et de les déclencher sur des stations de travail, aussi bien que dássurer l'échange d'information et la synchronisation entre les processus.

1. Wprowadzenie

Celem tego opracowania jest próba zbudowania uniwersalnego modelu środowiska przetwarzania rozproszonego opartego na lokalnej sieci komputerowej zarządzanej przez sieciowy system operacyjny NetWare firmy Novell. Przyjęto, że użytkownik powinien mieć możliwość oprogramowania takiego środowiska w standardowym języku wysokiego poziomu, rozszerzonym o funkcje biblioteczne pozwalające na tworzenie procesów potomnych, zdalne uruchamianie tych procesów, wymianę informacji między procesami i ich synchronizację. Pozwoli to na zachowanie elastyczności przy projektowaniu programów, a także umożliwi współbieżną (wykonywaną w tym samym czasie przez kilka komputerów) realizację niektórych jego fragmentów.

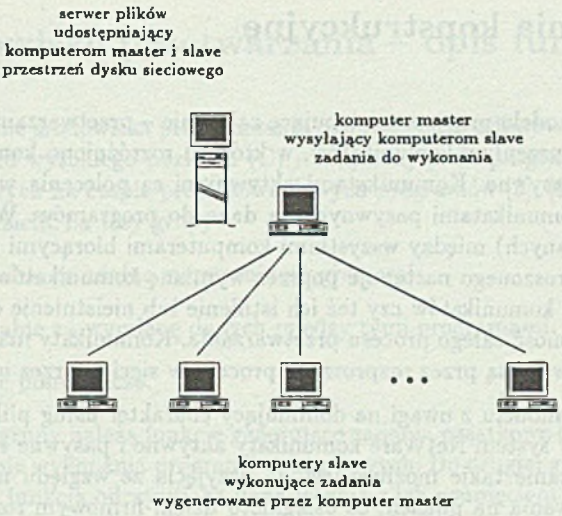
Przy projektowaniu środowiska wykorzystano trzy istotne cechy systemu sieciowego NetWare:

1. komputer nadrzędny – serwer i komputery podrzędne – stacje robocze mogą być przyłączone do wspólnego kabla transmisyjnego, po którym przesyłane są między nimi informacje,

2. stacje robocze mają zdolność do korzystania ze współdzielonej pamięci dyskowej serwera,

3. stacje robocze mają zdolność do wykonywania pobranych z serwera kopii programów we własnych pamięciach operacyjnych przez własne procesory.

Względy funkcjonalne budowanej maszyny wskazują na przyjęcie modelu przetwarzania opartego na farmie procesorów. W modelu tym ze zbioru stacji roboczych wybieramy komputer nadrzędny nadzorujący proces przetwarzania rozproszonego – master. Natomiast pozostałym komputerom przydzielamy funkcje komputerów podrzędnych – slave, które z kolei udostępniają komputerowi master swoją moc obliczeniową [21], [11]. Serwer plików, zarządzany przez sieciowy system operacyjny, wszystkim komputerom (master i slave) udostępnia swoją pamięć operacyjną oraz przestrzeń swojego dysku sieciowego (rys. 1). Komputer nadrzędny odpowiada za realizację programu głównego, a w tym m.in.



Rys. 1. Schemat powiązań między komputerami nadrzędnym, podrzędnymi i serwerem plików
Fig. 1. Scheme of dependencies between master slaves, and file server computers

za podział pierwotnego zadania na zadania cząstkowe, rozesłanie ich komputerom podrzędnym do wykonania, a następnie po ich wykonaniu, odebranie wyników cząstkowych i wygenerowanie wyniku globalnego. Generacja wyniku globalnego może być poprzedzona wieloma etapami generacji programów cząstkowych i odbierania wyników cząstkowych. Natomiast komputery podrzędne biernie oczekują na pojawienie się zadań cząstkowych, a po ich pojawieniu się wykonują je i generują wyniki cząstkowe [21], [11].

W ramach tak określonego modelu systemu sieciowego możemy określić drogę przepływu danych w tym środowisku między stacjami roboczymi a serwerem sieciowym [11]. I tak:

1. dane oraz kopie programu pobierane są z serwera do stacji roboczych,
2. następnie dane te przetwarzane są na stacjach roboczych,
3. po czym następuje odesłanie przetworzonych danych do serwera,
4. oraz przyjęcie przetworzonych danych i ewentualne ponowne udostępnienie ich stacjom roboczym.

2. Założenia konstrukcyjne

Przy budowie modelu przyjęto następujące założenie – przetwarzanie w maszynie zorientowane jest na przepływ komunikatów, w których rozróżniono komunikaty aktywne oraz komunikaty pasywne. Komunikatami aktywnymi są polecenia wykonania programów, natomiast komunikatami pasywnymi są dane do programów. Wymiana informacji (programów i danych) między wszystkimi komputerami biorącymi udział w procesie przetwarzania rozproszonego następuje poprzez wymianę komunikatów aktywnych i pasywnych. Przepływ komunikatów czy też ich istnienie lub nieistnienie determinuje realizowalność i skończoność całego procesu przetwarzania. Komunikaty mają swój określony format i identyfikowane są przez rozproszone procesy w sieci poprzez unikalne nazwy.

W omawianym modelu z uwagi na dominujący charakter usług plikowych świadczonych przez sieciowy system NetWare komunikaty aktywne i pasywne reprezentowane są jako pliki. Rozwiązanie takie możliwe jest do przyjęcia ze względu na bardzo wysoką efektywność operowania na plikach, co osiągnięto dzięki firmowym rozwiązaniom optymalizującym dostęp do dysku (takim jak przechowywanie najczęściej używanych plików oraz informacji o strukturze katalogów w pamięci podręcznej – cache, dostęp indeksowy do katalogów czy tzw. wyszukiwanie windowe [13], [14], [6]). W efekcie komunikacja i synchronizacja między stacjami roboczymi a serwerem nie odbywa się tylko i wyłącznie przez dysk sieciowy, a raczej przez szybszą pamięć operacyjną serwera.

Podobnie jak w procesie realizacji jakiegokolwiek algorytmu operuje się na zbiorze argumentów, a dokładniej na zbiorze danych (wejściowych i wyjściowych) i na zbiorze zmiennych (danych zmieniających swoje wartości w procesie przetwarzania), tak w zaproponowanym modelu operuje się faktycznie na zbiorze plików (komunikatów pasywnych), rozumianych także jako zbiór plików z danymi i zbiór plików ze zmiennymi. Sposób zdalnego wywoływania programów także realizowany jest poprzez pliki (komunikaty aktywne). Wówczas nazwy programów wraz z parametrami stanowią treść ogólnodostępnych plików. Komputery podrzędne, biorące udział w przetwarzaniu, oczekują na te pliki, a następnie po przejęciu ich na swój wyłączny użytek wykonują program zgodnie z ich treścią. Wszystkie pliki (z programami i z danymi) w procesie przetwarzania rozproszonego znajdują się w ogólnodostępnej przestrzeni dysku sieciowego na serwerze. Idea synchronizacji wymiany informacji poprzez pliki w środowisku sieciowego systemu operacyjnego Novell NetWare została najpierw opisana i wykorzystana w [21], a następnie także w [10] oraz [11].

Przez program realizowany w sposób rozproszony w przedstawionym środowisku przy powyższych założeniach będziemy rozumieli sekwencję generowania opisów programów cząstkowych, czyli komunikatów aktywnych, z zapewnieniem wymiany informacji między programami głównym a cząstkowymi poprzez komunikaty pasywne. Opisy programów cząstkowych generowane są przez program główny uruchomiony na komputerze master, natomiast ich realizacją zajmują się komputery slave.

3. Środowisko przetwarzania – opis funkcji

Za ustanowienie środowiska przetwarzania rozproszonego odpowiada zbiór funkcji napisanych w języku wysokiego poziomu (C) i dołączanych do programu głównego i programów cząstkowych na etapie projektowania tych programów [3], [1], [8], [17], [4].

Można je podzielić na trzy grupy:

1. odpowiedzialne za zdalne sterowanie programami,
2. odpowiedzialne za wymianę danych między tymi programami,
3. oraz funkcje pomocnicze.

Do pierwszej grupy należą funkcje generujące zadania cząstkowe oraz funkcje oczekujące na zakończenie wykonania programów cząstkowych. Do drugiej grupy należą: funkcja generująca dane, funkcja odczytująca dane łącznie z ich usunięciem oraz funkcja odczytująca dane i nie usuwająca ich. Trzecią grupę stanowią dwie funkcje pomocnicze – pierwsza pozwala na sprawdzenie liczby komputerów slave biorących udział w przetwarzaniu (co jest wykorzystywane w przypadku dedykowania określonych zadań konkretnym komputerom slave) oraz druga generuje sygnał zakończenia całego procesu przetwarzania, usuwając wszystkie chwilowo istniejące pliki robocze.

Przed uruchomieniem programu głównego na wszystkich komputerach slave należy uruchomić program `slave.exe`. Ustawia on te komputery w tryb oczekiwania na przyjęcie i wykonanie zadań cząstkowych, a po wykonaniu informuje komputer master o ich zakończeniu. Aby zsynchronizować wymianę zadań cząstkowych między komputerami master i slave, rozróżniono cztery stany, jakie mogą one przyjmować w procesie przetwarzania:

1. zadanie jest przygotowywane do wykonania,
2. zadanie jest gotowe do wykonania,
3. zadanie jest wykonywane,
4. zadanie zostało wykonane i można odebrać wyniki.

Dodatkowo każde zadanie cząstkowe rozpoznawane jest poprzez unikalny identyfikator. Zarówno stan zadania, jak i jego identyfikator zawarte są w nazwie pliku. Natomiast treścią tego pliku jest opis zadania cząstkowego, wygenerowany przez master do wykonania przez slave (mechanizm ten został opisany m.in. w [21], [11]).

Program uruchomiony na komputerach slave może pracować w jednym z dwóch trybów, dedykowanym i niededykowanym. W trybie dedykowanym każdemu komputerowi slave w momencie uruchamiania programu przypisywany jest unikalny numer porządkowy. Wówczas master może przesłać zadanie (a dokładniej opis tego zadania) konkretnemu komputerowi slave (np. ze względu na specyficzne właściwości tego komputera), przejmując niejako na siebie funkcję równoważenia obciążenia systemu generowanymi zadaniami cząstkowymi. Natomiast w trybie niededykowanym każdy komputer slave pracuje

na równych prawach, rywalizując z innymi komputerami slave o dostęp i przechwycenie na swój wyłączny użytek gotowych do wykonania zadań cząstkowych (a dokładniej opisów tych zadań). Wówczas system niejako automatycznie równoważy obciążenia komputerów slave. Zaleca się, aby w procesie przetwarzania rozproszonego wszystkie komputery slave pracowały albo w jednym, albo w drugim trybie. Wywołanie programu `slave.exe` z opcją `d` ustawia jego działanie w tryb dedykowany, natomiast z opcją `n` – w tryb niededykowany.

Opiszmy dokładnie wszystkie funkcje z poszczególnych grup, z ich nazwą, listą argumentów i wartościami, które zwracają.

Pierwszą grupę stanowią następujące funkcje:

- `void put_task ( char *identifier, char *assignment )` – funkcja ta tworzy DOS'owski plik z opisem zadania cząstkowego. Nazwa pliku składa się z dwóch części, pierwsza część odpowiada stanowi zadania (tzn. najpierw zadanie jest przygotowywane, a po przygotowaniu jest gotowe do wykonania), natomiast druga część odpowiada unikalnemu identyfikatorowi zadania. Identyfikator ten jest ciągiem znakowym (jego długość nie może przekraczać pięciu znaków) i podawany jest jako pierwszy parametr funkcji. Treść zadania cząstkowego stanowi całe polecenie DOS'owskie do wykonania przez komputer slave (tzn. nazwa programu i ewentualne parametry do programu) i podawana jest jako drugi parametr funkcji. Wywołanie w programie master następującej postaci: `put_task ( "aaaa1", "dir *.exe /w" );`, utworzy plik o nazwie `@@@aaaa1` i zapisze do niego polecenie `dir *.exe /w`, a następnie zmieni jego nazwę na `!@@aaaa1`, informując komputery slave o gotowości do wykonania programu zawartego w treści pliku. Funkcja ta wykorzystywana jest w trybie niededykowanej pracy komputerów slave.

- `void put_task_var_args ( char *identifier, char *assignment, ... )` – funkcja ta działa podobnie jak wcześniej opisana i różni się tylko sposobem wprowadzania argumentów. Dwa argumenty są obligatoryjne – pierwszy określa identyfikator zadania, a drugi nazwę programu wykonywalnego przez komputer slave, natomiast pozostałe argumenty są opcjonalne i określają parametry tego programu. Wywołanie w programie master następującej postaci: `put_task_var_args ( "aaaa1", "dir", "/a", "/w" );`, utworzy plik o nazwie `@@@aaaa1` i zapisze do niego polecenie `dir /a /w`, a następnie zmieni jego nazwę na `!@@aaaa1`, informując komputery slave o gotowości do wykonania programu zawartego w treści pliku. Funkcja ta wykorzystywana jest w trybie niededykowanej pracy komputerów slave.

- `void m_put_task_var_args ( char *identifier, char *assignment, ... )` – funkcja ta, podobnie jak dwie poprzednie, tworzy plik z opisem zadania cząstkowego do wykonania przez komputery slave. Przy czym zadanie nie jest jedno, lecz kilka i zakłada się, że wszystkie zadania wyszczególnione na liście parametrów funkcji wykonywane są dokładnie przez jeden komputer slave w kolejności ich wypisania (jeden parametr funkcji odpowiada tekstowi zawierającemu nazwę programu wykonywalnego wraz z parametrami). Wywołanie w programie master następującej postaci: `m_put_task_var_args ( "aaaa1", "dir /a", "dir /w" );`, utworzy plik wsadowy `aaaa1.bat` i zapisze do niego w kolejnych wierszach kolejne polecenia do wykonania umieszczone na liście, tzn. `dir /a` oraz `dir /w`. Następnie wywołanie

to utworzy plik opisu zadania do wykonania o nazwie @@@aaaa1 i zapisze do niego polecenie wykonania pliku wsadowego aaaa1.bat, a następnie zmieni jego nazwę na !@@aaaa1, informując komputery slave o gotowości do wykonania programu zawartego w treści tego pliku. Ten z komputerów slave, który przejmie plik !@@aaaa1, wykona polecenie aaaa1.bat, a przez to kolejne polecenia zawarte w pliku wsadowym. W poprzednio opisanych funkcjach generacja kilku plików opisu zadań wiązała się z kilkukrotnym wywołaniem tych funkcji, przy czym nie było gwarancji, że poszczególne zadania zostaną wykonane przez jeden komputer. Zastosowanie tej funkcji w pewnych sytuacjach może przyspieszyć wykonanie całego programu rozproszonego przez wyeliminowanie niepotrzebnego czekania. Funkcja ta, podobnie jak dwie wyżej opisane, jest wykorzystywana w trybie niededykowanej pracy komputerów slave.

Kolejne trzy funkcje w tej grupie są dokładnym odpowiednikiem trzech wyżej opisanych i ich działanie jest analogiczne. Stosowanie ich ma jednak sens tylko wtedy, gdy na komputerach slave uruchomiono program `slave.exe` w trybie dedykowanym. W sposobie wywoływania funkcje te różnią się tylko nazwą (do poprzednich nazw dodany jest przedrostek d_) oraz listą parametrów (dodany jest parametr, który determinuje numer komputera, na którym wykonane zostanie generowane zadanie: `int slave_number`).

Lista tych funkcji jest następująca:

- `void d_put_task ( int slave_number, char *identifier, char *assignment )`,
- `void d_put_task_var_args ( int slave_number, char *identifier, char *assignment, ... )`,
- `void d_m_put_task_var_args ( int slave_number, char *identifier, char *assignment, ... )`.

Dwie ostatnie funkcje w tej grupie pozwalają komputerom master na oczekiwanie na zakończenie wykonania programów cząstkowych przez komputery slave. Określone zadania identyfikowane są poprzez nazwy identyfikatorów, jakie zostały im przypisane podczas generowania. Zanim wyszczególnione zadanie (lub zadania) nie zostanie zakończone, program master nie podejmuje żadnej akcji.

Lista tych funkcji jest następująca:

- `void get_result ( char *identifier, ... )` – parametrami tej funkcji są identyfikatory oczekiwanych zadań, przy czym wystąpienie pierwszego parametru jest obligatoryjne, pozostałych jest opcjonalne. Gdy parametrów jest więcej niż jeden, to oczekiwane są wszystkie zadania umieszczone na liście, po czym następuje usunięcie wszystkich opisów oczekiwanych zadań. Funkcja ta powinna być wykorzystywana wówczas, gdy generacja zadań została wykonana przez funkcje, które w nazwie nie mają członu `m_` (np. `put_task()`).
- `void m_get_result ( char *identifier, ... )` – funkcja ta działa analogicznie jak opisana wyżej, tylko powinna być stosowana wówczas, gdy w nazwie funkcji generującej zadania był człon `m_` (np. `m_put_task_var_args()`).

Drugą grupę stanowią:

- `void put_data ( char *identifier, struct data_1 *ptr )` – funkcja ta tworzy DOS'owski plik z danymi zapisanymi binarnie. Nazwa pliku składa się z dwóch części, pierwsza część odpowiada stanowi przygotowywania pliku z danymi (tzn. najpierw dane są przygotowywane, a po przygotowaniu są gotowe do odczytania), natomiast druga część odpowiada unikalnemu identyfikatorowi danych. Identyfikator ten jest ciągiem znakowym (jego długość nie może przekraczać pięciu znaków) i podawany jest jako pierwszy parametr funkcji. Drugim parametrem funkcji jest wskaźnik do struktury przechowującej dane. Jeżeli zadeklarowana jest struktura: `struct data_1 { float f; int i; } *ptr;` i wykonane zostały następujące instrukcje przypisania: `ptr->f=1.2345; ptr->i=199;`, to wywołanie funkcji: `put_data ( "dddd1", ptr );`, utworzy plik o nazwie `@@@dddd1` i zapisze do niego binarnie dane zawarte w strukturze, a następnie zmieni jego nazwę na `-@@dddd1`, informując pozostałe komputery o możliwości ich odczytania. Może być ona wykorzystywana w trybie dedykowanej i niededykowanej pracy komputerów slave oraz może być wywoływana zarówno przez program główny, jak i programy cząstkowe.

- `struct data_1 *get_data ( char *identifier )` – funkcja ta czyta dane z pliku rozpoznawanego na podstawie identyfikatora wprowadzanego jako parametr tej funkcji. Gdy takiego pliku nie ma, to funkcja czeka, aż dane te pojawią się, wstrzymując dalsze wykonanie procesu. Może być ona przez to wykorzystywana do synchronizowania współbieżnie wykonywanych procesów. Gdy plik z danymi istnieje, to funkcja zwraca wskaźnik do struktury przechowującej odczytane dane. Jeżeli zadeklarowana jest struktura: `struct data_1 { float f; int i; } *ptr_out;` i wywołana została funkcja: `ptr_out = get_data ( "dddd1" );` i istniał właściwy plik z danymi, to w zmiennych `ptr_out->f` oraz `ptr_out->i` zachowane będą odpowiednio wartości: `1.2345` i `199`. Po odczytaniu danych funkcja usuwa plik, który te dane zawierał. Może być ona wykorzystywana w trybie dedykowanej i niededykowanej pracy komputerów slave oraz może być wywoływana zarówno przez program główny, jak i programy cząstkowe.

- `struct data_1 *read_data ( char *identifier )` – funkcja ta działa analogicznie jak poprzednia, z tą różnicą, że nie usuwa danych.

Trzecią grupę stanowią:

- `int check_no_of_slaves ()` – jest to bezparametrowa funkcja, która zwraca wartość odpowiadającą liczbie komputerów slave oczekujących na wykonanie zadań cząstkowych. Wykorzystywana jest praktycznie tylko wtedy, gdy komputery slave pracują w trybie dedykowanym. Pozwala ona wówczas określić maksymalną wartość parametru `int slave_number` dla tych funkcji, które generują zadania dedykowanym komputerom slave (np. `d_put_task ()`).

- `void terminate ()` – funkcja ta kończy proces przetwarzania rozproszonego i powinna być wywoływana na końcu programu głównego. Usuwa ona chwilowo istniejące pliki robocze, wykorzystywane np. do określenia maksymalnej liczby kompu-

terów slave biorących udział w przetwarzaniu (patrz poprzednia funkcja), a także generuje plik, który rozpoznawany przez programy `slave.exe`, kończy ich pracę.

Zilustrujmy sposób tworzenia rozproszonej aplikacji wykorzystującej wyżej opisane funkcje prostym przykładem [4]. Program uruchomiony na komputerze nadrzędnym (`example.exe`) powinien wysłać zbiór losowo wygenerowanych liczb całkowitych programom uruchomionym na komputerach slave (`total.exe`), które z kolei powinny obliczyć dla każdej otrzymanej liczby sumę od 1 do n (gdzie n jest wartością otrzymanej liczby) i odesłać ją komputerowi master. Program komputera master z kolei powinien zestawić wszystkie wyniki cząstkowe w jeden wynik globalny. Jeżeli spełnione zostaną warunki co do gotowości do przetwarzania odpowiedniej liczby komputerów slave (większej niż 1), to programy `total.exe` mogą być wykonywane współbieżnie.

Jedno z rozwiązań może wyglądać następująco:

```
// tekst zrodlowy programu example.cpp
#include <stdio.h>
#define MAX_INT 10
#define MAX_RANDOM_VALUE 100
struct data_1 { int a; } *var_str[ MAX_INT ], *var_out;
#include "bibliot.cpp" // biblioteka funkcji przetwarzania rozproszonego
void main() {
  int i, random_value; char *data_id, *task_id, *string; FILE *pointer;
  // inicjalizacja generatora
  randomize();
  // tworzenie danych wejsciowych i uruchamianie zadan na komputerach slave
  for ( i = 0; i < MAX_INT; i++ ) {
    if ((var_str[i] = (struct data_1*)malloc(sizeof(structdata_1))) == NULL) {
      cprintf ( "Not enough memory to allocate buffer\r\n" ); exit ( 0 ); }
    // tworzenie i przypisanie liczby pseudolosowej
    random_value = random ( MAX_RANDOM_VALUE );
    var_str [ i ] -> a = random_value;
    // tworzenie identyfikatora komunikatu pasywnego (in - dane we.)
    itoa ( i, string, 10 ); strcpy ( data_id, "in" ); strcat ( data_id, string );
    // tworzenie komunikatu pasywnego
    put_data ( data_id, var_str [ i ] );
    // tworzenie identyfikatora komunikatu aktywnego (ta - zadanie)
    strcpy ( task_id, "ta" ); strcat ( task_id, string );
    // tworzenie komunikatu aktywnego;
    // parametrem programu total wykonywanego przez slave jest
    // identyfikator komunikatu pasywnego
    put_task_var_args ( task_id, "total.exe", data_id, NULL );
  } // end for
  // odczytanie wynikow czastkowych i wygenerowanie globalnego zestawienia
  if ((var_out = (struct data_1*)malloc(sizeof(struct data_1))) == NULL) {
    cprintf ( "Not enough memory to allocate buffer\r\n" ); exit ( 0 ); }
  pointer = fopen ( "wynik.txt", "at" );
  for ( i = 0; i < MAX_INT; i++ ) {
    // tworzenie identyfikatora komunikatu pasywnego wyjsciowego (ou)
    itoa ( i, string, 10 ); strcpy ( data_id, "ou" ); strcat ( data_id, string );
    // funkcja get_data() czyta dane tylko wtedy gdy sa one dostepne
    var_out = get_data ( data_id );
    fprintf ( pointer, "%d ---> %d\n", var_str[ i ] -> a, var_out -> a );
```

```
    free ( var_str[ i ] );
  } // end for
  fclose ( pointer );
  free ( var_out );
  terminate ();
} // end main() "example.cpp"

// tekst zrodlowy programu total.cpp (na komputerach podrzednych
// program slave.exe zostal uruchomiony w trybie niededykowanym)
#include <stdio.h>
struct data_1 { int a; } *var_str;
#include "bibliot.cpp" // biblioteka funkcji przetwarzania rozproszonego
void main ( il, par )
int il; char *par[]; {
  int i; int sum = 0; char *out_id;
  if ((var_str = (struct data_1*)malloc(sizeof(struct data_1))) == NULL) {
    cprintf ( "Not enough memory to allocate buffer\r\n" ); exit ( 0 ); }
  var_str = get_data ( par [ 1 ] );
  for ( i = 1; i <= var_str -> a; i++ )
    sum += i;
  // utworzenie nowego identyfikatora (pozostawiajac poprzedni numer)
  strcpy ( out_id, par [ 1 ] ); out_id [ 0 ] = 'o'; out_id [ 1 ] = 'u';
  var_str -> a = sum;
  put_data ( out_id, var_str );
  free ( var_str );
} // end main() "total.cpp"
```

4. Podsumowanie

Przyjęte zasady działania i programowania maszyny poprzez tworzenie i wymianę komunikatów aktywnych i pasywnych pozwalają na rozpatrywanie rozpraszania na poziomie programów, nie określając jednak silnych powiązań między procesami, a tylko swobodne zależności między nimi. Jedynym kryterium powiązań komputerów nadrzędnych i podrzędnych jest dopasowanie identyfikatorów komunikatów. Koncepcja ta została w części przejęta z unix'owego systemu przetwarzania rozproszonego o nazwie Linda [17], [4], [18]. Jednak w tym przypadku została ona zaimplementowana w nowym środowisku – w sieci stacji roboczych pracujących pod systemem DOS zintegrowanych sieciowym systemem operacyjnym NetWare firmy Novell.

W przedstawionym modelu możliwe jest dziedziczenie funkcji komputera master przez komputery slave. Wówczas komunikaty aktywne powinny być generowane nie tylko przez komputery master, ale także przez komputery slave, co pozwala na tworzenie rozbudowanych konfiguracji systemów przetwarzania współbieżnego.

Budowa modelu, oparta na zbiorze funkcji operujących na komunikatach aktywnych i pasywnych, jest łatwo rozbudowywalna. I tak np. dzięki kilku nowym funkcjom system ten został rozbudowany o możliwość wysyłania dedykowanych zadań do wykonania przez konkretne komputery podrzędne.

W opracowaniu tym nie rozpatrywano problemów efektywnościowych związanych z obliczaniem w środowiskach rozproszonych w sieci, skupiając się tylko na sposobie oprogramowania tego środowiska. Zagadnienie zysków czasowych zostało opisane w [11] na przykładzie równoległych algorytmów sortowania [16].

Decydując się na projektowanie aplikacji uruchamianej w środowisku sieci stacji roboczych, należy zwrócić uwagę, iż w przypadku szybkich komputerów połączonych wolno pracującym systemem sieciowym (np. sieć oparta na standardzie Ethernet), przetwarzanie rozproszone w sieci polegające na wykonywaniu krótkich podzadań może nie spowodować skrócenia czasu liczenia, głównie z powodu zbyt dużych strat czasowych koniecznych do synchronizacji i komunikacji współbieżnie wykonywanych procesów. Jeżeli jednak zadanie daje się podzielić na spójne podzadania, a złożoność obliczeniowa podzadań jest stosunkowo duża, tak że czas ich realizacji jest niewspółmiernie duży do przesyłów sieciowych, to realizacja podzadań w sposób rozproszony w środowisku sieciowym daje istotne zyski czasowe.

LITERATURA

[1] Barkakati N.: The Waite Group's Turbo C++ Bible. Prentice Hall Computer Publishing, 1988.

[2] Ben-Ari M.: Podstawy programowania współbieżnego. WNT, Warszawa 1989.

[3] BORLAND C++ 3.0 Library Reference. Borland International, INC. 1991.

[4] Czajkowski G., Zieliński K.: Linda – środowisko do przetwarzania równoległego i rozproszonego w sieciach stacji roboczych. Informatyka nr 5, 1993.

[5] Czech Z.: Programowanie współbieżne, wybrane zagadnienia. Skrypt Politechniki Śląskiej nr 1777, Gliwice 1993.

[6] Day M., Neff K.: Troubleshooting NetWare for the 386. Prentice Hall International, England, 1991.

[7] GENESYS Manual. Transtech Parallel Systems Limited, 1991.

[8] Kernighan B. W., Ritchie D. M.: Język C. WNT, Warszawa 1988.

[9] Kozielski S., Szczerbiński Z.: Komputery równoległe. WNT, Warszawa 1993.

[10] Kozielski S., Tutajewicz R., Fagas R., Piec J.: Analiza możliwości i celowości wykorzystania sieci komputerowej do równoległej realizacji zadań wyszukiwania danych. Zeszyty Naukowe Politechniki Śląskiej, Seria: Informatyka z. 24, Nr kol. 1222, 1993.

[11] Krajewski R.: Przetwarzanie rozproszone w sieci Novell NetWare. Zeszyty Naukowe Politechniki Śląskiej, Seria: Informatyka z. 24, Nr kol. 1222, 1993.

[12] Milner R.: Communication and Concurrency. Prentice Hall International, England, 1989.

[13] Novell's Electronic Infobase of Technical Network Information.: Network Support Encyclopedia. Professional Volume, Novell, Inc., December 1991, 1994.

[14] NOVELL NetWare Version 3.11, System Documentation. Novell, Inc. 1991.

[15] Podraza R.: Sterowanie przepływowe. Informatyka nr 9-10, Warszawa 1988.

[16] Quinn M. J.: Designing efficient algorithms for parallel computers. McGram-Hill, New York 1987.

[17] Schoinas G.: (DRAFT) Issues on the implementation of PrOgramming SYstem for distriButed appLications (A free Linda implementation for Unix Networks). Department of Computer Science, University of Crete,

[18] Weiss Z., Gruźlewski T.: Programowanie współbieżne i rozproszone. WNT, Warszawa 1993.

[19] Węgrzyn S.: Systemy sterowania przepływem operacji i systemy sterowania przepływem argumentów. Zeszyty Naukowe Politechniki Śląskiej, Seria: Informatyka z. 24, Nr kol. 1222, 1993.

[20] Węgrzyn S., Czachórski T., Vidal P., Gille J.: Algorytmy i procesy równoległe w informatyce, w biologicznych systemach rozwojowych i w systemach masowej obsługi. Grant KBN 3 P 406 011 04.

[21] Wilk A.: Koncepcja wykorzystania sieci NetWare do realizacji przetwarzania równoległego. Zeszyty Naukowe Politechniki Śląskiej, Seria: Informatyka z. 26, Nr kol. 1222, 1994.

Recenzent: Prof. dr hab. inż. Andrzej Grzywak

Wpłynęło do Redakcji 20 września 1994 r.

Abstract

The goal of this paper is building a model of distributed processing environment, based on Novell NetWare local area network. The assumption was stated, user should write applications in such an environment in the high level language (C) extended by the special library functions, which give possibilities of generating child-processes, their remote call on the workstations and also exchange information between processes and processes synchronisation. The architecture based on client-server model is considered (fig. 1). As a method of synchronisation and communication between processes, file creating, renaming, and deleting was implemented. These special files are located on the file server and are accessible by all network processes. The files include distributed data and distributed commands to perform on the workstations, and are recognised by names. It allows to design distributed application on the higher level of peculiarity.