ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1993
Seria: INFORMATYKA z. 24 Nr kol. 1222

Rafał KRAJEWSKI

PRZETWARZANIE ROZPROSZONE W SIECI NOVELL NETWARE

Streszczenie. Artykuł naświetla problemy związane z możliwością wykorzystania lokalnej sieci komputerowej do przetwarzania rozproszonego. Przedstawiony jest algorytm sortowania rozproszonego oraz jego programowa implementacja w sieciowym systemie operacyjnym NetWare firmy Novell.

DISTRIBUTED PROCESSING ON NOVELL'S NETWARE NETWORK SYSTEM

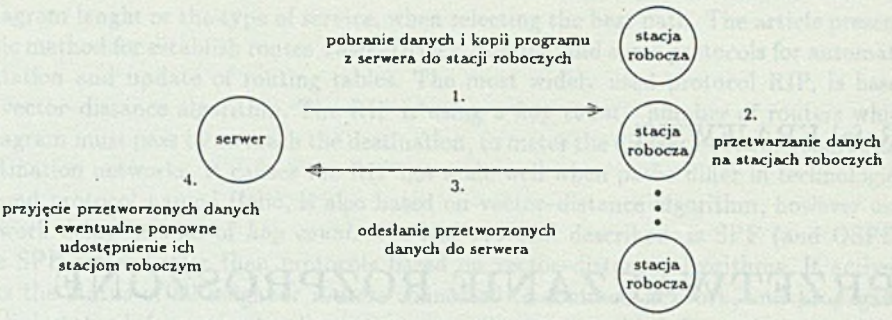
Summary. The article presents the problems of using local area network as a distributed processing system. Distributed algorithm of sorting and its programmable application on Novell's NetWare network system is presented.

TRAITEMENT DISTRIBUE SURE SYSTEMES RESEAUX NETWARE

Résumé. L'article prèsente les problèmes de l'utilisation des rèseaux locaux comme systèmes de traitement distribué. Un algorithme distribué de triage et son implementation logicielle sur le système reseau NetWare de Novell est présenté.

1. Wprowadzenie

Sieć komputerowa, rozumiana jako zbiór połączonych ze sobą komputerów wyposażonych w mechanizmy wymiany informacji zgodnie z określonymi protokołami komunikacyjnymi, umożliwia nie tylko proste i efektywne korzystanie ze wspólnych zasobów sieciowych, ale także pozwala wykorzystywać sumaryczną moc obliczeniową komputerów przyłączonych do sieci, stając się systemem przetwarzania rozproszonego. Przez przetwarzanie rozproszone w sieci rozumiemy możliwość podziału złożonego problemu na fragmenty, przesłanie tych fragmentów gotowym do przetwarzania współbieżnego (realizowanego w tym samym czasie) komputerom przyłączonym do sieci i odebranie od nich



Rys. 1. Droga przepływu danych przy przetwarzaniu rozproszonym w sieci
Fig. 1. Data flow during distributed processing in the network

wyników. Tak postrzegany sposób liczenia analogiczny jest do systemów transputerowych, gdzie liczeniem zadania zajmują się połączone ze sobą procesory (tramy). Pomimo bezsprzecznych zalet, jakie posiadają transputery, rozsądne i istotne wydaje się rozpatrywanie sieci komputerowej jako szeroko pojętego systemu pozwalającego realizować w sposób rozproszony złożone procesy obliczeniowe.

2. Charakterystyka sieciowego systemu Novell NetWare pod kątem wykorzystania do przetwarzania rozproszonego

Przy rozpatrywaniu problemu przetwarzania rozproszonego w sieciach ważne jest określenie cech danego systemu sieciowego pod względem architektury i rodzaju udostępnianych usług. System sieciowy NetWare (ver. 2.xx, 3.xx, 4) firmy Novell pozwala zbudować lokalną sieć komputerową o architekturze centralnej. Można w niej wyróżnić komputer nadrzędny - serwer oraz komputery podrzędne - stacje robocze. Serwer i stacje robocze przyłączone są za pośrednictwem kart sieciowych do wspólnego kabla transmisyjnego, po którym przesyłane są informacje. Na wszystkich komputerach przyłączonych do sieci uruchamiane jest oprogramowanie sieciowe pozwalające realizować funkcje komunikacyjne. Dodatkowo na serwerze uruchamiane są programy pozwalające udostępniać jego zasoby, z których dominującym jest wspólna przestrzeń pamięci dyskowej. Każda stacja robocza może korzystać ze współdzielonego dysku serwera. W omawianym systemie dwie stacje robocze bez dodatkowego oprogramowania nie mogą komunikować się między sobą. Natomiast programy pobrane przez stacje robocze z pamięci dyskowej serwera uruchamiane są na stacjach roboczych. Te dwie cechy systemu sieciowego NetWare, tzn. zdolność korzystania ze współdzielonej pamięci dyskowej i zdolność wykonywania pobranych z serwera programów w pamięci operacyjnej stacji roboczych, umożliwiają wykorzystanie go jako środowiska przetwarzania rozproszonego. Droga obiegu danych w omawianym środowisku przedstawiona jest na rys. 1.

3. Wielodostęp do zasobów dyskowych w sieci NetWare

Sieciowy system operacyjny Novell NetWare został zaprojektowany jako wielozadaniowy i wieloużytkownikowy. Oznacza to, że wielu użytkowników pracujących przy stacjach roboczych może korzystać z tych samych programów i z tych samych danych umieszczonych na serwerze.

W systemie operacyjnym NetWare isnieją cztery poziomy blokady dostępu do zbiorów znajdujących się na dysku sieciowym. Najniższym poziomem jest File Sharing Protocol Level One (określany w literaturze jako Automatic File Locking), który jest ustawiany przy rozpoczęciu pracy systemu. Na poziomie tym oprogramowanie systemowe serwera automatycznie blokuje zbiór na wyłączny użytek tej stacji, która odwołała się do niego jako pierwsza (np. program uruchomiony na stacji roboczej wykonał operację otwarcia zbioru). Blokada potrwa tak długo, jak długo trwać będzie odwołanie do tego zbioru, tzn. albo program uruchomiony na stacji roboczej zamknie go, albo zakończy swoje wykonanie. Program uruchomiony na stacji roboczej przy pierwszym poziomie blokady umożliwia odwołania do dysku sieciowego, tak jakby były to odwołania do dysku lokalnego. Może on wówczas bez żadnych zmian realizować operacje otwarcia zbioru, zamknięcia zbioru, czytania danych ze zbioru lub ich zapisywania. Jeżeli z danej stacji roboczej wystąpi próba otwarcia zbioru zajętego już przez program uruchomiony na innej stacji roboczej, to zostanie wygenerowany komunikat o wystąpieniu kolizji dostępu:

```
Network error: file in use during open.
File = <filename> Abort, Retry or Ignore?
```

Dalsza praca programu możliwa jest dopiero po interwencji użytkownika z zewnątrz. Za generowanie komunikatów o wystąpieniu ewentualnych kolizji odpowiada oprogramowanie sieciowe stacji roboczej.

Przy drugim poziomie blokady dostępu do zbiorów - File Sharing Protocol Level Two (określanym w literaturze jako Manual File Locking) - użytkownik sam kontroluje mechanizmy blokowania i odblokowywania dostępu do współdzielonych zbiorów. Na tym poziomie blokowania, program uruchamiany na stacji roboczej, chcący odwołać się do danego zbioru (np. wykonać operację zapisu), powinien najpierw zablokować go na swój wyłączny użytek, tak aby do momentu odblokowania nikt inny nie mógł z niego korzystać. Drugi poziom wykorzystywany jest w sytuacjach, gdy zmiany są dokonywane na dużych fragmentach jednego lub kilku zbiorów. Stosowany jest on także wtedy, gdy kontrolę nad przechwytywaniem komunikatów o ewentualnym zablokowanym dostępie powinna przejmować aplikacja uruchamiana na stacji roboczej, a nie systemowe oprogramowanie sieciowe tej stacji.

Trzeci poziom kontroli blokowania dostępu do współdzielonych zbiorów - File Sharing Protocol Level Three (określany w literaturze jako Logical File Locking), stosowany jest w przypadku programowych implementacji transakcji i odnosi się tylko do zbiorów, którym nadano atrybut *shareable*. Atrybut ten związany z danym zbiorem informuje sieciowy system operacyjny, że operacje blokowania i odblokowywania oraz synchronizacji dostępu do zbiorów przejmują na siebie aplikacje uruchamiane na stacjach roboczych,

zwalniając z tej funkcji system sieciowy. Na poziomie trzecim operacje blokowania, podobnie jak na poziomie drugim, dotyczą całych zbiorów.

Czwarty poziom kontroli blokowania dostępu - File Sharing Protocol Level Four (określany w literaturze jako Record Locking), związany jest z kontrolą blokowania dostępu nie do całych zbiorów, a tylko do tych ich części, które będą używane (np. modyfikowane) przez aplikację uruchomioną na danym komputerze, udostępniając innym część nie używaną (np. również do modyfikowania). Jest to najbardziej zaawansowany, dostarczający dużych możliwości i najbardziej elastyczny sposób synchronizacji dostępu do zbiorów na dysku sieciowym. W przypadku czwartego poziomu zbiorom, do ochrony których zamierzamy stosować mechanizm kontroli dostępu, powinno przypisać się atrybut *shareable* lub *shareable opened.*

Bez troski o szczegóły (ponieważ nie będzie to istotne dla przedstawionego przykładu) należy zaznaczyć, że implementacja programowa drugiego, trzeciego i czwartego poziomu blokady dostępu realizowana jest poprzez wywoływanie odpowiednich funkcji systemowych oprogramowania sieciowego NetWare [1].

Zbiory składowane na dysku sieciowym, do których możliwy jest współdzielony dostęp z poziomu innych stacji roboczych w tym samym czasie, można podzielić na dwie grupy. Do pierwszej grupy należą zbiory, które mogą być odczytywane przez wiele stacji, bez intencji zmiany ich zawartości. Są to głównie wykonywalne programy, nakładki programowe czy komunikaty o błędach (zbiorom takim można nadać znacznik *shareable read-only*). Do drugiej grupy należą zbiory zawierające dane, które mogą być zarówno czytane, jak i modyfikowane przez programy uruchamiane równolegle na wielu stacjach roboczych. Wówczas adaptacja sieciowa tych programów powinna uwzględniać aspekty odnoszące się do drugiego, trzeciego i czwartego poziomu kontroli dostępu.

4. Semafory w systemie NetWare

W sieciowym systemie operacyjnym Novell NetWare zostały zaimplementowane semafory, po pierwsze do ochrony zasobów, do których dostęp może odbywać się tylko na prawach wyłączności, po drugie jako mechanizm zabezpieczający aplikacje przed zbyt dużą i niedozwoloną licencyjnie liczbą użytkowników z niej korzystających [1].

Semafor S w systemie Novell NetWare określić można za pomocą dwójki NS i WS, gdzie NS jest nazwą semafora, a WS jest wartością semafora

$$S = (NS, WS)$$

Programowa implementacja semaforów została zrealizowana za pomocą funkcji systemowych oprogramowania sieciowego NetWare, identyfikowanych za pomocą szesnastkowego numeru funkcyjnego i osiągalnych z poziomu systemu operacyjnego DOS stacji roboczej poprzez przerwanie 21h.

Dostępne są następujące funkcje na semaforach:

1. Open a Semaphore - utwórz semafor.

2. Examine a Semaphore - testuj semafor.

3. Wait Semaphore - czekaj na semafor.

4. Signal a Semaphore - zwolnij semafor.

5. Close a Semaphore - usuń semafor.

Ponieważ w przedstawionym przykładzie (rozdz. 5) wykorzystane zostały tylko pierwsza, trzecia i czwarta funkcja, omówimy je bardziej szczegółowo, natomiast opis pozostałych znajduje się w [1].

AD 1. Funkcja Open a Semaphore - kod funkcji C5h (00h).

Zawartość rejestrów na wejściu:

```
AH        =  C5h
AL        =  00h
DS : DX   -  wskaźnik do ciągu reprezentującego nazwę semafora
CL        -  początkowa wartość semafora
```

Zawartość rejestrów na wyjściu:

```
AL        -  kod zakończenia wykonania funkcji
             =  00h  -  poprawne wywołanie funkcji
             =  FFh  -  niedozwolona wartość początkowa
             =  FEh  -  niedozwolona długość ciągu znakowego
             =  96h  -  brak pamięci roboczej w serwerze plików
BL        -  liczba stacji, które otworzyły dany semafor
CX, DX    -  systemowy numer semafora (semaphore handle)
```

Funkcja Open a Semaphore tworzy semafor, ustanawia jego nazwę i przypisuje tej nazwie wartość semafora. Wartość semafora jest ustawiana tylko raz, wtedy gdy wywoływana jest funkcja Open a Semaphore. Dopuszczalne wartości, które może on przyjmować przy otwieraniu, zawierają się w przedziale od 0 do 127. Początkowo ustawiona wartość może być zmieniona tylko poprzez wywołanie dwóch funkcji - Wait Semaphore i Signal a Semaphore. Po zmianach dopuszczalne wartości, jakie może przyjmować semafor, zawierają się w przedziale od -127 do 128.

W przypadku gdy semafor używany jest jako mechanizm zabezpieczający wywoływanie aplikacji przez zbyt dużą, niedozwoloną licencyjnie liczbę użytkowników, wykorzystywana jest tzw. liczba otwartych semaforów (open count). Oprogramowanie sieciowe wówczas przechowuje liczbę aplikacji korzystającej z danego semafora. Jeżeli aplikacja uruchamiana na danej stacji roboczej otwiera semafor o określonej nazwie, liczba otwartych semaforów zwiększa się o 1, jeżeli natomiast aplikacja zamyka dany semafor (poprzez wywołanie funkcji Close a Semaphore), wartość ta jest zmniejszana o 1. Jeżeli ostatnia aplikacja korzystająca z danego semafora kończy swoje wykonywanie i liczba aktualnie otwartych semaforów osiągnie zero, to semafor ten zostaje usunięty.

Semafory w systemie NetWare podczas wywoływania funkcji Open a Semaphore identyfikowane są poprzez nazwę, której długość może wynosić od 1 do 127 bajtów.

AD 3. Funkcja Wait Semaphore (decrement and wait if negative) - kod funkcji C5h (02h).

```
Zawartość rejestrów na wejściu:

   AH       = C5h
   AL       = 02h
   CX, DX   - systemowy numer semafora (semaphore handle)
   BP       - wartość reprezentująca czas czekania;
              podawana w 1/18 sekundy (00h - oznacza brak czekania)

Zawartość rejestrów na wyjściu:

   AL       - kod zakończenia wykonania funkcji
              = 00h - poprawne wywołanie funkcji
              = FEh - przekroczony czas oczekiwania
              = FFh - niedozwolony numer systemowy semafora
```

Funkcja Wait Semaphore jest stosowana wtedy, gdy semafory wykorzystywane są jako mechanizm gwarantujący wyłączny dostęp do zasobów. Wywołanie tej funkcji powoduje zmniejszenie o 1 wartości semafora, a następnie sprawdzenie, czy jest ona ujemna. Jeżeli tak, to proces realizowany na stacji roboczej czeka na semaforze przez czas określony w rejestrze BP. Jeżeli wyznaczony czas upłynie lub jeżeli semafor zostanie zwolniony (np. przez wywołanie funkcji Signal a Semaphore), to wartość związana z tym semaforem jest zwiększana o 1. Wynika z tego, że jeżeli wartość ta jest ujemna, to przed semaforem w kolejce znajdują się oczekujące zgłoszenia.

AD 4. Funkcja Signal a Semaphore (increment and release if waiting) - kod funkcji C5h (03h).

```
Zawartość rejestrów na wejściu:

   AH       = C5h
   AL       = 03h
   CX, DX   - numer systemowy semafora (semaphore handle)

Zawartość rejestrów na wyjściu:

   AL       - kod zakończenia wykonania funkcji
              = 00h - poprawne wywołanie funkcji
              = 01h - przepełnienie wartości semafora
              = FFh - niedozwolony numer systemowy semafora
```

Funkcja Signal a Semaphore, podobnie jak Open a Semaphore i Wait Semaphore, jest stosowana wtedy, gdy semafory wykorzystywane są jako mechanizm gwarantujący wyłączny dostęp do zasobów. Wywołanie tej funkcji powoduje zwiększenie o 1 wartości semafora i jeżeli jakikolwiek proces uruchomiony na stacji roboczej czekał na tym semaforze, to ten, który czekał najdłużej, zostanie wykonany.

5. Przykładowe rozwiązanie przetwarzania rozproszonego w sieci NetWare

Za przykład aplikacji wykonywanej w sposób rozproszony w sieci posłuży nam zadanie sortowania elementów.
Dany jest ciąg

$$a_1, a_2, a_3, \ldots a_n$$

Przez proces sortowania rozumiemy taką permutację elementów, która prowadzi do ich uporządkowania

$$a_{k_1}, a_{k_2}, a_{k_3}, \ldots a_{k_n}$$

że dla zadanej funkcji porządkującej $f()$, jest

$$f(a_{k_1}) \leq f(a_{k_2}) \leq f(a_{k_3}) \leq \ldots \leq f(a_{k_n}) \qquad (1)$$

Proponowany algorytm sortowania elementów w sposób rozproszony w sieci w przedstawionym przykładzie składa się z trzech etapów. Na pierwszym etapie procesy uruchomione współbieżnie na komputerach przyłączonych do sieci porcjują ciąg z elementami nieposortowanymi na odpowiednio małe podciągi. Na drugim etapie wykonywanym współbieżnie, realizowane są operacje sortowania elementów zawartych we wcześniej przygotowanych podciągach. Na trzecim i ostatnim etapie, także wykonywanym współbieżnie, realizowane jest scalanie posortowanych podciągów w jeden, całkowicie posortowany ciąg wynikowy.

Dane wejściowe znajdują się na dysku serwera w określonej kartotece w postaci zbioru o określonej nazwie i o określonym rozmiarze. Program realizujący poszczególne etapy przetwarzania także znajduje się na dysku sieciowym w tej samej kartotece. Każda ze stacji roboczych biorąca udział w przetwarzaniu rozproszonym i mająca dostęp do wspólnej kartoteki z programem i danymi, wykonuje ten sam proces poprzez pobranie dostępnej wszystkim kopii programu do swojej pamięci operacyjnej. W podanym przykładzie sortowania rozproszonego wykorzystano pierwszy poziom blokady dostępu do zbiorów i funkcje semaforowe oprogramowania sieciowego NetWare.

W czasie realizacji sortowania rozproszonego wprowadzono system nazewnictwa zbiorów umożliwiający bezpośrednie stwierdzenie stanu zbioru - do porcjowania, do sortowania czy do scalania oraz jego lokalizację w całym zbiorze wynikowym. I tak dla nazwy składającej się z ośmiu znaków (zgodnie z konwencją przyjętą w systemie DOS, nie uwzględniając trzech znaków rozszerzenia), pierwszy znak informował wszystkie procesy, że jest to zbiór roboczy biorący udział w procesie przetwarzania rozproszonego. Drugi znak w nazwie zbioru informował procesy, czy jest on przed, w trakcie, czy po poszczególnych etapach przetwarzania. Natomiast ostatnie sześć znaków w nazwie zarezerwowane było na numer pozwalający poszczególnym procesom umiejscowić dany zbiór w zbiorze wynikowym.

Etap 1:

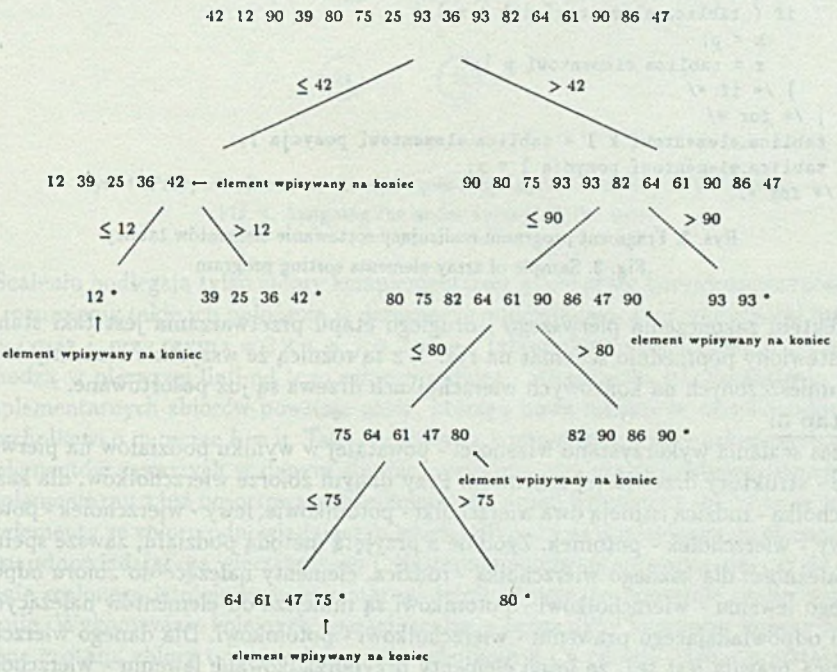
Wszystkie procesy uruchomione na stacjach roboczych rywalizują o dostęp do zbioru

z danymi (pierwszą nazwą zbioru z nieposortowanymi elementami jest "!!1"). Próba dostępu realizowana jest poprzez wywoływanie funkcji `findfirst()` w języku Borland C++ (funkcja ta wykorzystuje DOS'owskie przerwanie 0x4E [2]). Odpowiada ona za szukanie w kartotece zbioru o nazwie "!!*", gdzie '*' zgodnie z konwencją przyjętą w systemie DOS, zastępuje wszystkie znaki. Ta stacja, która znajdzie zbiór jako pierwsza, w celu przejęcia go na swój wyłączny użytek zmienia jego nazwę na unikatową (w tym konkretnym przypadku na "!#1"), tak aby inne stacje tego samego zbioru w tym samym czasie już nie znalazły. Aby wyeliminować możliwość znalezienia zbioru i zmiany jego nazwy przez więcej niż jedną stację, co prowadziłoby do błędnego przetwarzania, operacja znajdowania zbioru i operacja zmiany jego nazwy realizowane były jako krytyczne i zostały zabezpieczone funkcjami semaforowymi. I tak początkowo wszystkie procesy wywołały funkcję Open a Semaphore z nazwą „pierwszy" i z wartością 1 (przypisanie wartości semaforowi zostało zarejestrowane w systemie NetWare tylko przy pierwszym wywołaniu tej funkcji; zob. rozdz. 4). Następnie przed wywołaniem funkcji znajdowania zbioru i zmiany jego nazwy, każdy z procesów wywołał funkcję Wait Semaphore dla tego samego semafora. Ten z procesów, który zrobił to jako pierwszy, zmniejszył jego wartość o 1 i ponieważ nie była ona ujemna przeszedł dalej. Każdy następny proces po wywołaniu funkcji Wait Semaphore zmniejszył aktualną wartość semafora o 1 i ponieważ była ona już ujemna, musiał czekać na semaforze aż do jego zwolnienia. Po znalezieniu zbioru i po zmianie jego nazwy, semafor jest zwalniany poprzez wywołanie funkcji Signal a Semaphore, tak aby następny, najdłużej oczekujący w kolejce proces, mógł rozpocząć szukanie. Taką metodę szukania kolejnych zbiorów stosuje każdy z procesów biorących udział w przetwarzaniu.

Kolejną czynnością na pierwszym etapie przetwarzania rozproszonego jest sprawdzanie rozmiaru zbioru przez ten z procesów, który przejął go na swój wyłączny użytek. Jeżeli zbiór jest dostatecznie duży, to zmieniana jest ponownie jego nazwa na unikatową, odpowiadającą pierwszemu etapowi (w tym konkretnym przypadku na "!~1") i realizowana jest operacja porcjowania zbioru na dwa, przy czym do pierwszego zbioru wpisywane są wszystkie elementy mniejsze lub równe od pierwszego elementu, a do drugiego zbioru wpisywane są wszystkie elementy większe od pierwszego elementu (pierwszy z tych dwóch zbiorów nazywa się "!~2", natomiast drugi "!~3"). W wyniku porcjowania zbiorów, ich wzajemne rozmieszczenie może zostać opisane za pomocą drzewa. Stąd utworzonym zbiorom przyporządkowano numery zgodnie z następującą regułą: zbiór odpowiadający wierzchołkowi w drzewie o numerze n dzieli się na lewego potomka o numerze $2 \times n$ i prawego potomka o numerze $2 \times n + 1$. Przyporządkowanie zaczynamy od wierzchołka o numerze 1. Po wpisaniu wszystkich elementów do nowych zbiorów, zmieniane są ich nazwy na takie, które informują procesy w sieci, że mogą one być ponownie przechwytywane do dalszego przetwarzania (w naszym przypadku jest to: "!!2" i "!!3"). Wartość graniczna rozmiaru zbioru decydująca o tym, czy powinien on być porcjowany, czy nie, w przedstawionym przykładzie została wyznaczona doświadczalnie i równa jest w przybliżeniu 5% początkowej wartości, tzn. dla zbioru zawierającego początkowo 20 000 elementów, którego rozmiar wynosi około 100 000 bajtów, jako wartość graniczną przyjęto 5000 bajtów. Wpisywanie wszystkich elementów do dwóch nowo utworzonych zbiorów realizowane jest w takiej kolejności, w jakiej następuje odczytywanie z wcześniej przejętego zbioru z jednym wyjątkiem. Pierwszy element z przejętego zbioru, względem którego dzieli się pozostałe elementy, nie jest wpisywany jako pierwszy element do pierwszego zbioru, ale jako ostatni

element do pierwszego zbioru. Ma to na celu eliminację następującego przypadku. Podzielone zbiory po utworzeniu i nadaniu im odpowiednich nazw zaczynają istnieć w kartotece na dysku sieciowym na takich samych prawach, jak pierwotny zbiór z danymi wejściowymi. O zbiory te rozpoczyna się rywalizacja o przejęcie na własny użytek przez procesy przetwarzania rozproszonego w sieci. Podobnie po przejęciu tych zbiorów, sprawdzany jest ich rozmiar i jeżeli są większe od pierwotnie przyjętej wartości, następuje porcjowanie. Gdyby więc pierwszy element rozstrzygający o przynależności pozostałych elementów do poszczególnych zbiorów był ponownie pierwszy, podziały byłyby silnie niezrównoważone. Wstawienie natomiast pierwszego elementu na koniec zbioru gwarantuje, że o podziale decydować będą za każdym razem inne elementy, co z kolei pozwala na lepsze zrównoważenie rozmiarów zbiorów przygotowywanych do sortowania.

Zilustrujmy na schemacie (rys. 2) kolejne porcjowania realizowane przez komputery przyłączone do sieci. Dany jest zbiór zawierający 16 elementów i załóżmy, że porcjowanie dokonywane jest zawsze wtedy, gdy w zbiorze są ponad 4 elementy.



Rys. 2. Porcjowanie zbiorów (* koniec porcjowania; liczba elementów ≤ 4)
Fig. 2. Files portioning (* end of portioning; number of elements ≤ 4)

Etap 2:

W przypadku, gdy proces wykonywany na stacji roboczej przechwyci zbiór, którego rozmiar jest mniejszy od wyznaczonej granicy, rozpoczyna się drugi etap przetwarzania rozproszonego. Podobnie jak na pierwszym etapie, zmieniana jest nazwa przechwyconego zbioru z "!!numer" poprzez "!#numer" na "!_numer", gdzie ta ostatnia informuje wszystkie procesy w sieci, że elementy zawarte w tym zbiorze są właśnie sortowane. Operacje szukania zbioru oraz po znalezieniu, zmiana jego nazwy, podobnie jak na pierwszym etapie, chronione są semaforem. W przedstawionym rozwiązaniu wykorzystano sortowanie przez proste wybieranie. W metodzie tej najpierw z wszystkich elementów od 1 do n wybieramy najmniejszy, następnie wybieramy najmniejszy spośród pozostałych $n-1$, potem $n-2$ itd., aż dojdziemy do ostatniego, posortowanego elementu. Fragment programu pozwalający zrealizować przedstawiony algorytm, może wyglądać następująco [4]

```
int      x, tablica_elementow[ MAX_LB_ELEMENTOW ];
int      p, pozycja, k;
int      n;
/* n - wartosc ostatniego indeksu w tablicy liczac od 0 */
for ( pozycja = 0; pozycja <= n - 1; pozycja++ ) {
   k = pozycja;
   x = tablica_elementow[ pozycja ];
   for ( p = pozycja + 1; p <= n; p++ ) {
      if ( tablica_elementow[ p ] < x ) {
         k = p;
         x = tablica_elementow[ p ];
      } /* if */
   } /* for */
   tablica_elementow[ k ] = tablica_elementow[ pozycja ];
   tablica_elementow[ pozycja ] = x;
} /* for */
```

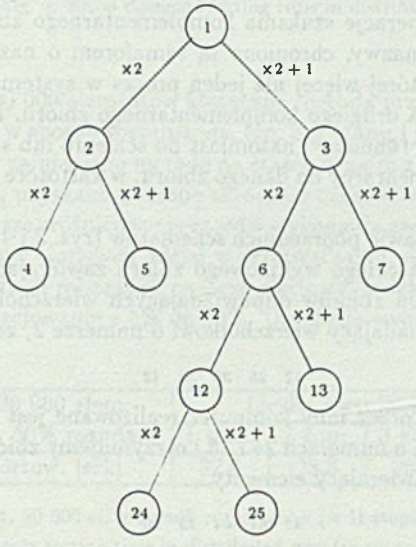
Rys. 3. Fragment programu realizujący sortowanie elementów tablicy
Fig. 3. Sample of array elements sorting program

Efektem zakończenia pierwszego i drugiego etapu przetwarzania jest taki stan, jak przedstawiony poprzednio schemat na rys. 2, z tą różnicą że wszystkie elementy w zbiorach umieszczonych na końcowych wierzchołkach drzewa są już posortowane.

Etap 3:

Podczas scalania wykorzystano własności - powstałej w wyniku podziałów na pierwszym etapie - struktury drzewiastej schematu. Przy danym zbiorze wierzchołków, dla każdego wierzchołka - rodzica istnieją dwa wierzchołki - potomkowie, lewy - wierzchołek - potomek i prawy - wierzchołek - potomek. Zgodnie z przyjętą metodą podziału, zawsze spełniona jest zależność: dla każdego wierzchołka - rodzica, elementy należące do zbioru odpowiadającego lewemu - wierzchołkowi - potomkowi są mniejsze od elementów należących do zbioru odpowiadającego prawemu - wierzchołkowi - potomkowi. Dla danego wierzchołka - rodzica prawdą jest też, że jeżeli elementy przyporządkowane lewemu - wierzchołkowi - potomkowi są mniejsze od elementów przyporządkowanych prawemu - wierzchołkowi - potomkowi, to nie ma w tym drzewie ani jednego końcowego wierzchołka, które miałoby większe elementy od elementów lewego - wierzchołka - potomka i jednocześnie mniejsze elementy od elementów prawego - wierzchołka - potomka.

Z uwagi na powyższe własności, proces scalania jest bardzo podobny do procesu dzielenia, przy czym odwrócona jest kolejność postępowania. Nadając poszczególnym wierzchołkom w drzewie kolejne numery zgodnie z przyjętą regułą dla przedstawionego na poprzednim schemacie drzewa (rys. 2), mamy



Rys. 4. Przyporządkowanie numerów poszczególnym wierzchołkom w drzewie
Fig. 4. Assigning the nodes number in the tree

Scaleniu podlegają tylko zbiory komplementarne, gdzie przez komplementarność zbiorów rozumiemy takie ich położenie w drzewie, że odpowiadające im wierzchołki mają numery i oraz j, przy czym $i = 2 \times n$, a $j = 2 \times n + 1$ (zbiory komplementarne to takie, które pochodzą w pierwszej linii od tego samego rodzica - wierzchołka n). Po scaleniu dwóch komplementarnych zbiorów powstaje zbiór, którego nowe miejsce w drzewie odpowiada wierzchołkowi o numerze $k = n$. Tak więc każdy z komputerów, który zakończył sortowanie elementów zawartych w danym zbiorze, sprawdza, czy dla tego zbioru istnieje zbiór komplementarny z już posortowanymi elementami. Jeżeli istnieje, to zbiory te są scalane, tzn. elementy ze zbioru odpowiadającego wierzchołkowi j są dopisywane do elementów ze zbioru odpowiadającego wierzchołkowi i. Następnie ponownie sprawdza się, czy do zbioru właśnie scalonego istnieje komplementarny. Jeżeli istnieje, to ponownie realizowane jest scalanie. Wykonywanie kolejnych operacji scalania przez dany komputer kończy się, gdy scalone zostaną zbiory odpowiadające wierzchołkom 2 i 3 (dzięki czemu powstaje zbiór wynikowy odpowiadający wierzchołkowi 1) albo gdy do aktualnie posortowanego lub do aktualnie scalonego nie istnieje w danej chwili komplementarny. Wówczas komputer ten przechodzi do etapu I lub II, ewentualnie po wykryciu zbioru z wynikiem kończy przetwarzanie. Czynności scalania mogą być wykonywane równolegle przez wszystkie komputery

biorące udział w przetwarzaniu. Jeżeli jeden z komputerów, który nie może w danej chwili znaleźć drugiego z komplementarnych zbiorów, zakończy etap scalania, wtedy inny komputer po utworzeniu tego zbioru jest w stanie je scalać.

Może się zdarzyć że jeden z komputerów utworzy zbiór lewy - potomek i będzie czekał na prawy komplementarny. Po pewnym czasie inny komputer mógł utworzyć ten właśnie zbiór prawy - potomek i będzie czekał na lewy komplementarny, co mogłoby prowadzić do wzajemnej blokady. Operacje szukania komplementarnego zbioru oraz w przypadku znalezienia, zmiana jego nazwy, chronione są semaforem o nazwie „drugi". Eliminuje się przez to sytuację, w której więcej niż jeden proces w systemie w tym samym czasie realizuje operację szukania drugiego komplementarnego zbioru. Podczas scalania nazwa zbioru jest zmieniana na "!@numer", natomiast po scaleniu lub stwierdzeniu, że w danej chwili nie istnieje komplementarny do danego zbioru, w kartotece pozostawiany jest zbiór o nazwie "!$numer".

Przedstawmy na podstawie poprzednich schematów (rys. 2 i 4) kolejne kroki scalania, prowadzące do utworzenia całego wynikowego zbioru zawierającego wszystkie posortowane elementy. Po scaleniu zbiorów odpowiadających wierzchołkom o numerach 4 i 5 otrzymujemy zbiór odpowiadający wierzchołkowi o numerze 2, zawierający elementy

12 25 36 39 42

W tym samym czasie przez inny komputer realizowane jest scalanie zbiorów odpowiadających wierzchołkom o numerach 24 i 25 i otrzymujemy zbiór odpowiadający wierzchołkowi o numerze 12, zawierający elementy

47 61 64 75 80

oraz dla następnych kroków scalania

12	⊕	13	⇒	6:	47	61	64	75	80	82	86	90	90						
6	⊕	7	⇒	3:	47	61	64	75	80	82	86	90	90	93	93				
2	⊕	3	⇒	1:	12	25	36	39	42	47	61	64	75	80	82	86	90	90	93	93
scalane wierzchołki					zawartość zbiorów otrzymanych w wyniku scalania														

otrzymujemy zbiór wynikowy, w którym elementy spełniają warunek (1).

6. Podsumowanie

Przedstawiona aplikacja przetwarzania rozproszonego w sieci wykorzystana została do sortowania 20 000 elementów. Kluczami były losowo wygenerowane liczby całkowite z przedziału od 0 do 20 000. Wartość graniczna, informująca procesy uruchomione na komputerach w sieci, czy przechwycony zbiór należy jeszcze dzielić, czy już sortować, została ustalona doświadczalnie na 5% początkowego rozmiaru zbioru. Testy realizowane były na nieobciążonej i odizolowanej lokalnej sieci ethernetowskiej, do której przyłączone były komputery ALR Business VEISA 80386/33 MHz (zgodne z IBM PC) z systemem operacyjnym DOS (ver. $\geq$ 3.30), jako stacje robocze, oraz COMPAQ ProSignia 80486/33MHz (zgodny z IBM PC) z sieciowym systemem operacyjnym Novell NetWare 3.11, jako serwer. Czasy sortowania w sposób rozproszony w sieci w funkcji wzrastającej liczby komputerów przedstawiono na rys. 5.

Sortowanie 20 000 elem.	Liczba sortujących komputerów				
granica porcj. - 5% rozmiaru	1 kom.	2 kom.	3 kom.	4 kom.	5 kom.
całk. czas sortow. [sek]	123.2	75.4	63.4	65.8	79.4

Rys. 5. Czasy sortowania 20000 elementów w sposób rozproszony
Fig. 5. 20000 elements sorting time in distributed way

Czas sortowania 20 000 elementów klasyczną metodą przez proste wybieranie na jednym komputerze (nie w sposób rozproszony, tzn. bez etapu I i III) wynosi około 13 minut.

W przypadku gdy zamieniono metodę na etapie drugim z sortowania przez proste wybieranie na quicksort, uzyskano znaczne skrócenie czasów wykonania zadań. Przy czym zaobserwowano mniejsze zróżnicowanie czasów wykonania zadań w zależności od wzrastającej liczby komputerów biorących udział w przetwarzaniu. Aby zminimalizować operacje przesyłu zbiorów ze stacji roboczych na dysk sieciowy, zwiększono wartość granicy, przy której realizowano porcjowanie z 5% do 50%. Uzyskane wyniki przedstawiono na rys. 6.

Sortowanie 20 000 elem.	Liczba sortujących komputerów				
granica porcj. - 50% rozmiaru	1 kom.	2 kom.	3 kom.	4 kom.	5 kom.
całk. czas sortow. [sek]	32.6	28.3	26.8	27.9	31.4

Rys. 6. Czasy sort. 20 000 el. w sposób rozproszony (w II etapie zastosowano quicksort)
Fig. 6. 20 000 elements sorting time in distributed way (in second part quicksort was used)

Czas sortowania 20 000 elementów metodą quicksort [4] nie w sposób rozproszony (bez etapu I i III) wynosi 10 sekund tzn. najkrócej ze wszystkich czasów uzyskanych w testach.

Formułując wnioski dotyczące przedstawionego sposobu przetwarzania rozproszonego w sieci NetWare można stwierdzić, że:

- przyjęty algorytm sortowania umożliwia równomierne rozłożenie procesów obliczeniowych wśród komputerów przyłączonych do sieci, co w efekcie pozwala na:
  - — eliminację do minimum okresów czekania komputerów na przetwarzanie,
  - — skrócenie czasu wykonania całego zadania,
  - — pełniejsze wykorzystanie komputerów w sieci,
  - — zwiększenie wydajności systemu sieciowego,
- udało się istotnie zmniejszyć czasy sortowania przez zwiększanie liczby komputerów tylko w przypadku, gdy rozpatrywaną metodą w etapie II było sortowanie przez proste wybieranie (patrz rys. 5), (sortowanie przez proste wybieranie charakteryzuje się złożonością obliczeniową rzędu n^2, biorąc pod uwagę porównywanie elementów jako operację dominującą),

- w przypadku gdy zastosowano quicksort na II etapie sortowania rozproszonego, nie udało się zmniejszyć czasów sortowania przez zwiększanie liczby komputerów biorących udział w przetwarzaniu: przede wszystkim z powodu strat czasowych spowodowanych przesyłami zbiorów między stacjami roboczymi a serwerem oraz zastosowaniem mechanizmów synchronizacji (sortowanie metodą quicksort charakteryzuje się złożonością obliczeniową rzędu $n \times \log n$, biorąc pod uwagę porównywanie elementów jako operację dominującą),

- przedstawiony system przetwarzania rozproszonego, w którym działania podejmowane są w zależności od istnienia określonych danych, jest systemem sterowania przepływem danych.

Celem artykułu nie było podanie nowej metody sortowania rozproszonego (przedstawiona metoda jest znana w literaturze jako tzw. równoległy quicksort i została opisana m.in. w [7] dla komputera ze wspólną pamięcią). Chodziło głównie o zwrócenie uwagi, 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.

Opracowanie to jest wynikiem prac prowadzonych na zajęciach laboratoryjnych w IITiS PAN w Gliwicach w ramach studium podyplomowego nt. możliwości przetwarzania rozproszonego w systemie sieciowym Novell NetWare.

LITERATURA

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

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

[3] S. Węgrzyn, *Podstawy Informatyki*, PWN, Warszawa 1982.

[4] N. Wirth, *Algorytmy + Struktury Danych = Programy*, WNT, Warszawa 1989.

[5] *GENESYS Manual*, Transtech Parallel Systems Limited, 1991.

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

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

Recenzent: Prof. dr hab. inż. Zbigniew Czech

Wpłynęło do Redakcji dnia 21.09.1993

Abstract

The computer network is not only the data exchange system to use common network recourses, but also the environment that can use single computers linked together to produce more powerful processing tool. Concept of network distributed processing is as follows: complex problem is divided into many sub problems, and all sub problems are computed by several processors in parallel way. In this article network distributed processing system was implemented on the Novell's NetWare network platform. Novell NetWare (ver. 3.11) is a server - client network, with common server's disk resource access to the workstations connected to the same cable system (fig. 1). During processing all programs are taken from the common server's disk by workstations and computed by their processors. It is very important for distributed application to consider all network aspects, such as multiple access to the same programs or data, using semaphores or blocking and releasing file access functions. As an example, distributed algorithm of sorting on the network is presented. It consists of three parts: portioning of input files (fig. 2), sorting small pieces of files (fig. 3) and consolidation of small files into one output file with sorted elements (fig. 4). All parts may be performed in parallel way by the workstations. Communication and synchronisation problem was solved by creating temporary work files with specific names. They are interpreted by the network processes and the proper action is applied. Also NetWare semaphore system functions were used. The sorting time when number of computers were changed is shown on fig. 5 and 6.