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

Leszek KOTZIAN

OBLICZENIA RÓWNOLEGŁE W SIECI Z PROTOKOŁEM TCP/IP

Streszczenie. Artykuł przedstawia środowisko umożliwiające wykonywanie równoległych obliczeń w sieci z protokołem TCP/IP. Zaproponowane rozwiązanie wykorzystuje protokół RPC oraz mechanizmy komunikacji między procesami. Implementacja środowiska została wykonana w języku C i testowana na stacjach roboczych Sun.

PARALLEL COMPUTATIONS ON TCP/IP PROTOCOL NETWORK

Summary. The paper presents an environment for parallel computations on TCP/IP network. This method is based on RPC protocol and Inter-Process Communication mechanizms. It was implemented in C language and tested on Sun workstations.

CALCULS EN PARALLELE SUR RESEAUX AU PROTOCOLE TCP/IP

Resume. L'article presente une methode de calculs en parallele sur de reseaux TCP/IP. La methode utilise le protocole RPC at les functions de communication entre les processus. La methode a ete implementee en langage L et testee sur de stations de travail Sun.

1. Wstęp

Poniższe opracowanie przedstawia narzędzia umożliwiające wykonywanie obliczeń równoległych w środowisku Unix. Podstawowym założeniem wybranego rozwiązania jest użycie RPC (*Remote Procedure Call*) jako sposobu komunikowania się między procesami na różnych komputerach. To, w jaki sposób wykorzystano RPC, określiło użycie mechanizmów komunikacji między procesami, a są nimi komunikaty i semafory. Realizacja funkcji RPC przebiega synchronicznie, tzn. program, z którego jest wywołana dana funkcja, czeka na jej zakończenie. Taki sposób komunikacji wymaga dodatkowych funkcji i programów zapewniających "równoległość" oraz obsługę takich zdarzeń jak rozpoczęcie i zakończenie obliczeń przez inny komputer.

W przyjętym rozwiązaniu prowadzenie równoległych obliczeń polega na wykonywaniu programów przez komputery połączone w sieci - dalej zwane *oddalonymi*. Napisy (nazwa programu wraz z parametrami) rozsyłane są przez jeden komputer zwany dalej *centralnym*. W proponowanym rozwiązaniu po wykonaniu zadania(programu) zwracana jest jedynie informacja o jego zakończeniu. Wprowadzenie zmian obejmujących uwzględnienie innych typów danych wejściowych i wyjściowych oraz samego sposobu wykonania zadania wymagałoby modyfikacji niektórych istniejących już funkcji. Programy i funkcje zostały napisane w języku C, a uruchamiane na stacjach roboczych Sun z systemem operacyjnym SunOS 4.1.x i 5.3 (Solaris 2.3) połączonych w sieci Ethernet.

2. Idea działania

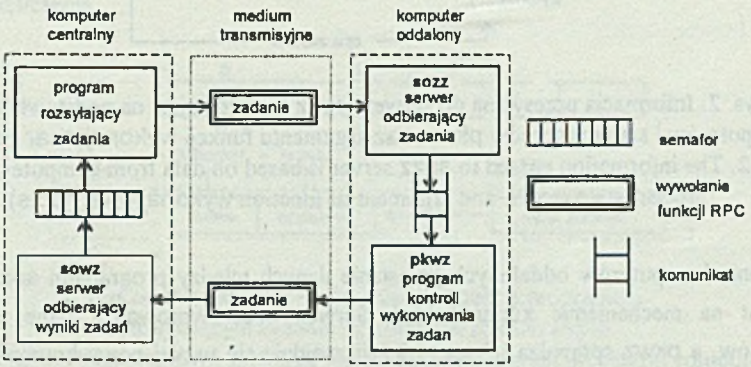
Przedstawione narzędzia (funkcje i programy) realizują obliczenia równolegle na podstawie modelu *klient-serwer(y)*. Klientem jest proces (na komputerze centralnym) rozsyłający zadania, a serwerem lub serwerami procesy wykonujące poszczególne zadania (na komputerach oddalonych). Do kierowania równoległymi obliczeniami służy kilka funkcji wywoływanych z poziomu programu użytkownika. Funkcje te mają na celu sprawdzenie połączenia z oddalonymi komputerami, przesłanie, oczekiwanie na zakończenie zadania oraz zakończenie przetwarzania równoległego.

Program, który rozsyła zadania do obliczeń, wywołuje zdalną funkcję wykonywaną przez serwer RPC sozz uruchomiony na jednym z komputerów oddalonych. Funkcja zwraca jedynie

wynik przesłania danych do programu kontroli wykonania zadania pkwz (który również znajduje się na komputerze oddalonym). Po wykonaniu obliczeń wysyłany jest komunikat (również jako wywołanie zdalnej funkcji) do serwera RPC sowz uruchamianego na komputerze centralnym. Serwer odbierający wyniki zrealizowanych zadań przekazuje informacje o ich wykonaniu do programu rozsyłającego zadania.

3. Opis rozwiązania

Wykonywanie równoległe obliczeń możliwe jest po uprzednim przygotowaniu środowiska. Polega to na uruchomieniu programu serwera sozz (z kartoteki, w której musi znajdować się program kontroli wykonywania zadań pkwz na komputerach oddalonych) i stworzeniu zbioru o nazwie SERVER zawierającego nazwy tych komputerów. Na centralnym komputerze w kartotece, w której uruchamiany jest program rozsyłający zadania, należy również umieścić program serwera RPC sowz (rys. 1).

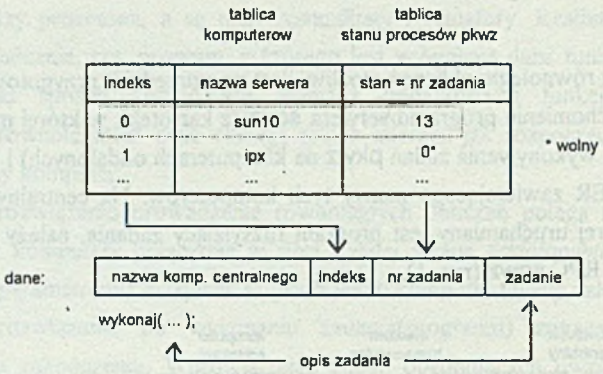


Rys. 1. Model realizujący rozsyłanie i kontrolę wykonania zadań
Fig. 1. The task distribution and control model

Przed wysłaniem pierwszego zadania sprawdzane jest połączenie z serwerami RPC sozz uruchomionymi na komputerach oddalonych (wyszczególnionych w zbiorze SERVER), uruchomienie programów pkwz) i według tego tworzona jest tablica komputerów biorących udział w obliczeniach. Aktualizowane są również tablice zawierające stan procesu pkwz (wolny lub zajęty - wykonujący zadanie) i numer zadania (identyfikujący zadanie wykonywane

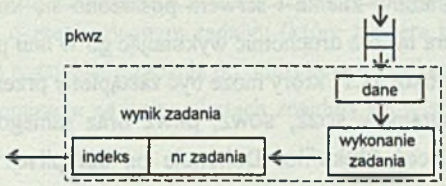
przez dany komputer). Również na wstępie są inicjowane semafory po stronie komputera centralnego, którego użycie będzie omówione w dalszej części.

Program wysyłający zadanie (teraz oprócz opisu zadania przesyłana informacja zawiera nazwę komputera centralnego, numer zadania i indeks tablicy serwerów, pod którym znajduje się nazwa oddalonego komputera) przesyła je do oddalonego komputera poprzez wywołanie funkcji serwera RPC sozz (rys. 2). Serwer sozz przekazuje zadanie programowi kontroli wykonania zadań i zwraca wynik wykonanej operacji jako powrót z funkcji RPC.



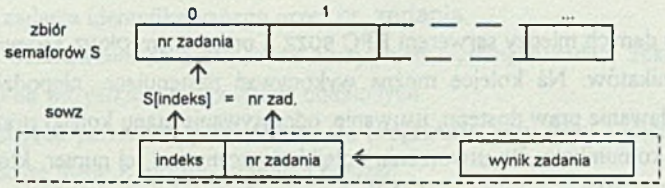
Rys. 2. Informacja przesyłana do serwera sozz tworzona jest na podstawie tablic komputerów i stanu procesów pkwz oraz argumentu funkcji wykonaj(char *opis)
Fig. 2. The information passed to sozz server is based on data from computer, and pkwz process state arrays, and argument of function wykonaj(char *opis)

Po stronie komputerów oddalonych przesłanie danych między programem sozz a pkwz oparte jest na mechanizmie komunikatów. Serwer sozz wprowadza dane do kolejki komunikatów, a pkwz sprawdza jej stan i jeżeli znajdzie się w niej nowy komunikat, to go odczyta. Następnie pkwz oddziela informacje identyfikujące zadanie od jego treści i wykonuje je (rys. 3). Zgłoszenie o zakończeniu wykonywania zadania jest przesyłane wraz z numerem zadania i indeksem tablicy serwerów do serwera sowz (na komputerze centralnym), znów jako wywołanie funkcji RPC.



Rys. 3. Wykonanie zadania i przesłanie wyniku przez program kontroli wykonania zadania pkwz
Fig. 3. Program pkwz executes task and sends back results

Serwer RPC sowz komunikuje się z programem rozsyłającym zadania poprzez semafory. Kolejne numery semaforów w zbiorze odpowiadają indeksom tablicy komputerów. Przez wykonanie na semaforze *S* operacji *VG(S,n)* (gdzie *n* równa się numerowi zadania) sowz informuje o wykonaniu zadania *n* przez dany komputer oddalony (rys. 4). Przez ten czas program rozsyłający zadania może wysyłać kolejne zadanie, czekać na wysłanie (kiedy wszystkie serwery są zajęte), czekać na wykonanie konkretnego zadania lub zakończyć równoległe obliczenia.



Rys. 4. Przesyłanie wyniku między sowz a programem rozsyłającym zadania na komputerze centralnym
Fig. 4. Program sowz forwards results to user program on central computer

4. Użyte mechanizmy służące do komunikacji i synchronizacji między procesami

4.1. RPC

Jak już wspomniano, RPC zostało użyte do komunikacji pomiędzy procesami na różnych komputerach.

Przy tworzeniu programów klienta i serwera posłużono się kompilatorem rpcgen. Tak powstały program serwera można uruchomić wykonując go w linii poleceń. Jako transport dla RPC został wybrany protokół TCP, który może być zastąpiony przez UDP.

Do generowania programów sozz, sowz, pkwz oraz samego programu rozsyłającego zadania służy zbiór poleceń Makefile. Dokonuje on kompilacji i scalania odpowiednich zbiorów. Cały zestaw składa się z następujących zbiorów:

stale.c semafor.c komunikat.c prz.c
sowz.c pkwz.c sozz.c
zad.x odp.x test.c

z których po prawidłowym wykonaniu Makefile powinny powstać:

test pkwz sozz sowz.

W omawianym rozwiązaniu istnieją dwa serwery: jeden uruchamiany na oddalonym komputerze sozz odbierający dane i przekazujący je do programu pkwz oraz drugi sowz - odbierający zgłoszenia o wykonaniu zadań - uruchamiany na komputerze centralnym, z którego są wysyłane zadania.

4.2. Komunikaty i semafory

Przesyłanie danych między serwerem RPC sozz a programem pkwz zapewnia mechanizm kolejki komunikatów. Na kolejce można wykonywać następujące, niepodzielne operacje: tworzenie, nadawanie praw dostępu, usuwanie, odczytywanie stanu kolejki oraz zapisywanie i odczytywanie komunikatu. Po utworzeniu kolejki zwracany jest jej numer, który identyfikuje ją w systemie. Wykorzystywana kolejka przez funkcje serwera i program kontroli wykonywania zadań jest jednokierunkowa (komunikaty wysyłane są jedynie przez serwer).

Informacje o wykonanych zadaniach rejestruje serwer sowz nadając wartości odpowiednim semaforom. Stan semaforów odczytywany jest przez program rozsyłający zadania. Semafory są pogrupowane w zbiory (w szczególnym przypadku jest to tylko jeden semafor). Cały zbiór rozróżniany jest przez swój identyfikator, semafory w zbiorze również posiadają unikalne numery. Na semaforach można wykonywać następujące, niepodzielne operacje: tworzenie, nadawanie praw dostępu, usuwanie, odczytywanie stanu oraz testowanie semafora. W przypadku omawianej realizacji tworzony jest jeden zbiór semaforów, którego liczebność odpowiada liczbie komputerów oddalonych. Kolejne semafory odpowiadają indeksom tablicy serwerów. Wartość danego semafora może być równa zero (zadanie jeszcze nie wykonane) lub numerowi zadania (zadanie wykonane). Wartości semaforom nadaje funkcja serwera sowz

po odebraniu zgłoszenia o zrealizowanym zadaniu (które zawiera indeks tablicy serwerów i numer zadania). Program rozsyłający zadania po odczytaniu semafora - zeruje go.

Funkcje wykonujące operacje na komunikatach znajdują się w zbiorze komunikat.c, a na semaforach - semafor.c.

5. Objaśnienie funkcji służących do równoległego wykonywania obliczeń

Sekwencję instrukcji przetwarzania równoległego rozpoczyna funkcja int pocz() sprawdzająca uruchomione serwery na komputerach oddalonych, inicjująca semafor, tablicę komputerów i stanu. Jeśli środowisko jest w pełni przygotowane, można korzystać z pozostałych funkcji, a są nimi:

int wykonaj(char *opis) - wysyła zadanie do obliczeń, którego opis stanowi opis. Zwraca numer zadania.

int czekaj_zad(int nr_zadania) - wstrzymuje wykonanie programu do momentu wykonania zadania identyfikowanego przez nr_zadania.

koniec() - oczekiwanie na zakończenie wszystkich wysłanych zadań, zakończenie pracy równoległej na wszystkich komputerach oddalonych.

Stałe określające parametry początkowe dla programów serwerów i klienta mają swoje definicje w zbiorze stale.c, przedstawionym poniżej.

```
#define MAXS   10   /* maks. ilosc komp. oddalonych i semaforow w zbiorze */
#define MAXNZ  4    /* maks. dl. pola numeru zadania i stanu w tablicy komputerow */
#define MAXNS  20   /* maks. dl. nazwy komputera */
#define EUS    50   /* maks. ilosc zbiorow semaforow */
#define QNM    1    /* ilosc kanalow komunikatow */
#define MAXK   100  /* maks. ilosc sprawdzanych kanalow */
```

Poprzez modyfikacje tych stałych można np. zmieniać długość opisu zadania, liczbę komputerów biorących udział w obliczeniach równoległych. Po dokonaniu zmian w zbiorze stale.c należy ponownie utworzyć programy serwerów i klienta posługując się poleceniami ze zbioru Makefile.

Dodatkowa uwaga dotyczy obsługi sygnałów mogących przerwać równoległe wykonywanie obliczeń przed zakończeniem funkcji koniec(). Aby obsłużyć takie zdarzenia, należy zdefiniować sygnały (np. wciśnięcie klawiszy ctrl-C, zerwanie łączności z terminalem),

których przechwycenie spowoduje zatrzymanie programów pkwz. Jest to realizowane przez funkcję fine() kończącą pracę programów pkwz na serwerach i przykładową sekwencję:

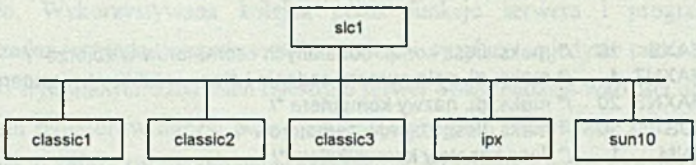
```
#include<signal.h>
#include"prz.c"
..
..
(void)signal(SIGTERM,fine);          /* zakonczenie programowe */
(void)signal(SIGQUIT,fine);          /* przerwanie: ctrl-C */
(void)signal(SIGHUP,fine);       /* zerwanie lacznosci z terminalem sterujacym */
```

umieszczoną na początku programu. Funkcje służące do równoległego wykonywania obliczeń znajdują się w zbiorze prz.c.

6. Przykład obliczeń równoległych

Pierwszy przykład został zaczerpnięty z opracowania [4]. Obliczenia polegają na kompilacji i scalaniu modułów programu napisanego w języku C. W tym przypadku kompilacja poszczególnych modułów może być wykonywana przez komputery oddalone, a scalanie uzależnione jest od zakończenia kompilacji.

Obliczenia przeprowadzono na stacjach roboczych Sun współpracujących za pomocą połączeń typu Ethernet (rys. 5). Moc obliczeniowa komputerów była zróżnicowana.



Rys. 5. Konfiguracja sieci lokalnej wykorzystywanej w obliczeniach
Fig. 5. The scheme of LAN used during experiments

Komputerem centralnym został slc1. Zbiór SERVER zawierający nazwy komputerów oddalonych wyglądał następująco (znaki występujące po # są traktowane jako komentarz):

```
# nazwa_komp   wersja_SunOS      model              cpu
#
sun10        # 4.1.3            SPARCstation-10    SuperSPARC Model 30
```

```
classic1        # 4.1.3C      SPARCclassic      50 MHz microSPARC
classic2
classic3
ipx
```

Kompilowane moduły znajdowały się w kartotekach, z których uruchamiano programy serwerów RPC sozz na komputerach oddalonych. Tekst programu rozsyłającego zadania był następujący:

```
#include<stdio.h>
#include"prz.c"
#include<signal.h>

void main()
{
 int i;

(void)signal(SIGTERM,fine);
(void)signal(SIGINT,fine);
(void)signal(SIGQUIT,fine);
(void)signal(SIGHUP,fine);

i=pocz();
if (il=0)
 {
 wykonaj("cc -c -DHAVE_UNISTD_H=1 -DDIRENT=1 -DHAVE_ALLOCA_H=1 gzip.c");
 wykonaj("cc -c -DHAVE_UNISTD_H=1 -DDIRENT=1 -DHAVE_ALLOCA_H=1 zip.c");
 wykonaj("cc -c -DHAVE_UNISTD_H=1 -DDIRENT=1 -DHAVE_ALLOCA_H=1 deflate.c");
 wykonaj("cc -c -DHAVE_UNISTD_H=1 -DDIRENT=1 -DHAVE_ALLOCA_H=1 trees.c");
 wykonaj("cc -c -DHAVE_UNISTD_H=1 -DDIRENT=1 -DHAVE_ALLOCA_H=1 bits.c");
 wykonaj("cc -c -DHAVE_UNISTD_H=1 -DDIRENT=1 -DHAVE_ALLOCA_H=1 unzip.c");
 wykonaj("cc -c -DHAVE_UNISTD_H=1 -DDIRENT=1 -DHAVE_ALLOCA_H=1 inflate.c");
 wykonaj("cc -c -DHAVE_UNISTD_H=1 -DDIRENT=1 -DHAVE_ALLOCA_H=1 util.c");
 wykonaj("cc -c -DHAVE_UNISTD_H=1 -DDIRENT=1 -DHAVE_ALLOCA_H=1 crypt.c");
 wykonaj("cc -c -DHAVE_UNISTD_H=1 -DDIRENT=1 -DHAVE_ALLOCA_H=1 lzw.c");
 wykonaj("cc -c -DHAVE_UNISTD_H=1 -DDIRENT=1 -DHAVE_ALLOCA_H=1 unlzw.c");
 wykonaj("cc -c -DHAVE_UNISTD_H=1 -DDIRENT=1 -DHAVE_ALLOCA_H=1 unpack.c");
 wykonaj("cc -c -DHAVE_UNISTD_H=1 -DDIRENT=1 -DHAVE_ALLOCA_H=1 unlzh.c");

 i=wykonaj("cc -c -DHAVE_UNISTD_H=1 -DDIRENT=1 -DHAVE_ALLOCA_H=1 getopt.c");

 czekaj_zad(i);
 wykonaj("cc -o gzip gzip.o zip.o deflate.o trees.o bits.o unzip.o
     inflate.o util.o crypt.o lzw.o unlzw.o unpack.o
     unlzh.o getopt.o");
 koniec();
 }
return;
}
```

Dokonane pomiary czasu generowania programu wykonywalne dla różnej ilości komputerów oddalonych przedstawia tabela 1. Czas wykonania tego zadania przez najszybszy z Sun'ów sun10 wynosi 24 [s]. Przyśpieszanie jest rozumiane jako stosunek czasu wykonania zadania sekwencyjnie do czasu wykonania go równolegle.

Tabela 1

Czasy kompilacji i scalania modułów programu w zależności od liczby komputerów oddalonych

Komputer centralny	Komputery oddalone	Czas wykonania zadania [s]	Przyśpieszenie
sic1	sun10, classic1-3, ipx	8	3.0
sic1	sun10, classic1-3	9	2.6
sic1	sun10, classic1-	10	2.4
sic1	sun10, classic1	13	2.0

Jak widać z pomiarów, czas wykonania zadania maleje wraz z liczbą stacji oddalonych uczestniczących w obliczeniach.

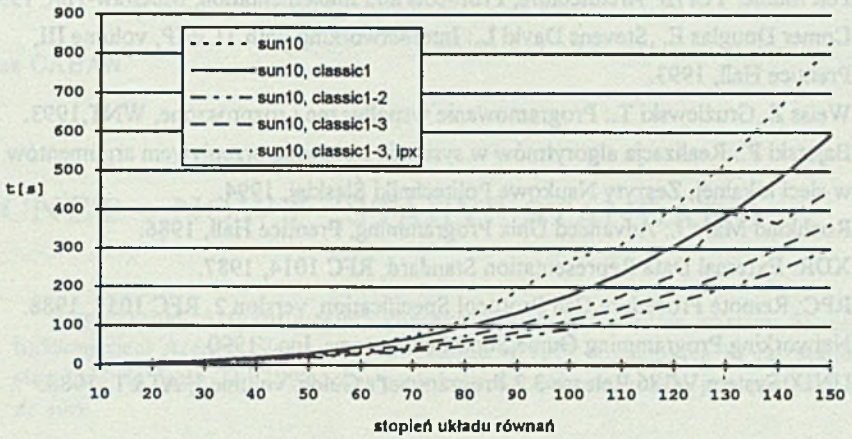
Drugi przykład dotyczył rozwiązywania układów równań liniowych metodą wyznaczników. Zadanie polegało na obliczeniu wyznacznika układu równań, a następnie na obliczaniu kolejnych pierwiastków, które to obliczenia mogły być wykonywane na komputerach oddalonych. Wykonano je w tej samej konfiguracji przy zmieniającym się stopniu układu równań i ilości komputerów oddalonych (tabela 2).

Tabela 2

Czas rozwiązania układu równań w zależności od stopnia układu równań i ilości komputerów oddalonych ([s]).

Komputery oddalone	Stopień układu równań														
	10	20	30	40	50	60	70	80	90	100	110	120	130	140	150
sun10	1	2	6	14	28	48	78	117	192	271	311	408	525	664	826
sun10, classic1	3	6	12	19	31	48	70	99	134	179	236	308	384	480	593
sun10, classic1-2	3	5	10	16	25	38	53	76	103	142	178	233	395	366	452
sun10, classic1-3	3	6	9	14	21	32	44	63	86	115	147	187	236	303	363
sun10, classic1-3, ipx	3	5	10	12	18	28	37	51	72	94	122	160	200	246	300

Wykres zależności czasu rozwiązania układu równań od stopnia układu i liczby komputerów oddalonych



Na podstawie pomiarów z przykładu drugiego widać, że większe przyśpieszenie uzyskuje się podczas realizacji zadań składających się z obliczeń o dużej złożoności (długi czas wykonania pojedynczego obliczenia). Zadania składające się z obliczeń o mniejszej złożoności (w omawianym przykładzie są to rozwiązania układu równań o liczbie niewiadomych mniejszej od 60) wykonywane są szybciej w sposób sekwencyjny lub przy mniejszej liczbie komputerów oddalonych.

7. Uwagi końcowe

Przedstawione środowisko służy do realizacji obliczeń równoległych w sieci TCP/IP. Wykonano również pomiary czasu obliczeń przykładowych zadań. Podsumowując, można stwierdzić, że osiągnięte rezultaty potwierdzają przydatność systemu do rozpraszania obliczeń w sieci komputerowej.

LITERATURA

[1] Feit Sidnie: TCP/IP Architecture, Protocols and Implementation, McGraw-Hill, 1993.
[2] Comer Douglas E. ,Stevens David L.: Internetworking with TCP/IP, volume III, Prentice Hall, 1993.
[3] Weiss Z.,Gruźlewski T.: Programowanie współbieżne i rozproszone, WNT,1993.
[4] Bajerski P.: Realizacja algorytmów w systemie sterownia przepływem argumentów w sieci lokalnej, Zeszyty Naukowe Politechniki Śląskiej, 1994.
[5] Rochkind Marc J.: Advanced Unix Programming, Prentice Hall, 1986.
[6] XDR: External Data Representation Standard, RFC 1014, 1987.
[7] RPC: Remote Procedure Call Protocol Specification, version 2, RFC 1057, 1988.
[8] Networking Programming Guide,Sun Microsystems, Inc., 1990.
[9] UNIX System V/386 Release 3.2 Programmer's Guide, volume I, AT&T, 1988.

Recenzent: Dr inż. Krzysztof Nałęcki

Wpłynęło do Redakcji 28 lipca 1994 r.

Abstract

The article presents method of performing parallel computations in TCP/IP network. The method is based on RPC protocol and IPC mechanizms like semaphores and messages. RPC is a high-level communications paradigm which allows to design programs within client-server network model. With RPC, the client makes a procedure call which sends request to the server. When the server performs request the client suspends its execution. Together with added mechanizms RPC can distribute tasks and colect results of the computations.

Such a method was implemented and tested. The presented functions can be performed on the network according to the program prepared by a user.